COMPRESSIONION OF PULSE CODE MODULATION SIGNAL USING TYPE IV DISCRETE COSINE TRANSFORM BASED ON THE MP3 MODEL.

BY:

ANDREW YOTUI CHEPYEGON (B.Sc)

REG NO I56/CE/11000/2006

A Thesis Submitted in Partial Fulfillment of the Requirements for the Award of the Degree of Master of Science (Electronics and Instrumentation) in the School of Pure and Applied Sciences of Kenyatta University.

March 2017

DECLARATION

This study is my original work and has not been presented for the award of a degree or any other award in any other University.

Andrew Yotui Chepyegon
Department of Physics
Kenyatta University

Signature…….. Date…….

This thesis has been submitted with our approval as University supervisors.

University Supervisors

Dr Willis Ambusso Signature……. Date…….
Department of Physics
Kenyatta University

Dr Mathew Munji Signature……. Date…….
Department of Physics
Kenyatta University

DEDICATION

I dedicate this Thesis to the Supreme God, the Almighty and the giver of Life, Knowledge and Wisdom, my strong pillar, and the source of my inspiration. I also dedicate this work to my Wife Mariam, who incessantly prayed for my success during my Thesis development; My Parents who patiently kept their hopes in me alive. They were my encouragement all the way. I dedicate this Thesis also to my brothers and sisters who have been so understanding even as they looked up to me as their eldest brother, and whose encouragement made sure that I gave it all it took to finish that which I had started. To my Workmates both at St. Paul's University and Institute of Advanced Technology who have been supportive in every way possible. Thank you all and may God bless you.

ACKNOWLEDGEMENT

I thank my project Supervisors, Dr Willis Ambusso and Dr Mathew Munji for their guidance patience and encouragement during the unenviable period of developing this project. Appreciation also goes to the Institute of Advanced Technology (IAT) for allowing me time off to concentrate on generating code and creating the write ups for this project. It was not easy working and being a student at the same time. Finally I thank all those relatives, friends and colleagues who encouraged me to press on even when my resources were running low.

ACRONYMS AND ABBREVIATIONS

AIFF	Audio Interchange File Format
CD	Compact Disc
DBA	Digital Broadcast Audio
DCT	Discrete Cosine Transform
FFT	Fast Fourier Transform
IMDCT	Inverse MDCT
JPEG	Joint Photograph Expert Group
MDCT	Modified Discrete Cosine Transform
MDST	Modified Discrete Sine Transform
MP3	MPEG-1 Audio Layer III
MPEG	Motion Picture Expert Group
MUSICAM	Masking Pattern Adapted Universal Subband Integrated Coding and Multiplexing
ODG	Objective Difference Grade
PEAQ	Perceptual Evaluation of Audio Quality
PCM	Pulse Code Modulation
PQF	Polyphase Quadrature Filter.
QMF	Quadrature Mirror Filter
RIFF	Resource Interchange File Format
SDG	Subjective Difference Grade
SNR	Signal to Noise Ratio
TDAC	Time domain aliasing cancellation
WAV	Waveform

TABLE OF CONTENTS

DECLARATION ii
DEDICATION iii
ACKNOWLEDGEMENT iv
ACRONYMS AND ABBREVIATIONS v
LIST OF TABLES viii
LIST OF FIGURES ix
LIST OF LISTINGS x
ABSTRACT xi
CHAPTER ONE 1
INTRODUCTION 1
1.1 Background to the study 1
1.2 Theoretical Framework 2
1.3 Statement of the research problem 6
1.4 Objectives 7
1.4.1 General Objective 7
1.4.2 Specific Objectives 7
1.5 Rationale/Justification of the Study 7
CHAPTER TWO 8
LITERATURE REVIEW 8
2.1 Introduction 8
2.2 Lossless and Lossy coders 9
2.3 Existing Coders and Standards 10
2.3.1 Subband Coders 10
2.3.2 MPEG-1, Layer I and II 12
2.3.3 Wavelet Coders 12
2.4 Modified Discrete Cosine Transform (MDCT) 13
2.5 Perceptual Evaluation of Audio Quality (PEAQ) 14
CHAPTER THREE 17

RESEARCH METHODOLOGY 17
3.1 Coding: 17
3.2 Algorithm: 18
3.3 Algorithm description and Analysis 18
3.3.1 PCM Signal capture 20
3.3.2 MP3 audio encoding 23
3.3.3 MDCT 24
3.3.4 Compressed signal output formatting 29
3.3.5 Frame Format 35
3.3.5.1 Header 39
3.3.5.2 Side Information 39
3.3.5.3 Main Data 43
3.3.5.4 Ancillary Data 44
CHAPTER FOUR 45
RESULTS AND DISCUSSION 45
4.1 The Interface 45
4.1.1 File Open/Capture 46
4.1.2 Sampling Rate 47
4.1.3 Compression Type 47
4.1.4 File Manipulation 47
4.1.5 Miscellaneous 48
4.2 Quantitative Analysis 48
4.3 Qualitative Analysis 52
CHAPTER FIVE 64
CONCLUSIONS AND RECOMMENDATIONS 64
5.1 Conclusions 64
5.2 Recommendation for future work 65
REFERENCES 66
APPENDICES 69
The Code 69

LIST OF TABLES

Table 4.1: Compression percentages for audio signals at a sampling rate of 32000Hz ... 49

Table 4.2: Compression percentages for audio signals at a sampling rate of 41000Hz ... 49

Table 4.3: Compression percentages for audio signals at a sampling rate of 48000Hz ... 50

Table 4.4: Compression ratios for MP1, MP2 and MP3 compression techniques for audio signals at a sampling rate of 44100Hz 51

Table 4.5: Listening Tests Grading Scale based on ITU-R BS.1284 standard ranging from 1.0 to 5.0. 53

Table 4.6: Listener satisfaction survey questionnaire. 54

Table 4.7: Table showing comparrisons between the test results of the study, with the ITU-R BS.1284 grading system. 54

Table 4.8: SDG description..... 59

Table 4.9: Objective Difference Grade (ODG) scores for ten audio samples sampled at 48 kHz, 16-bit PCM 60

Table 4.91: ODG values and file size for four codecs on most common bit rates. 61

LIST OF FIGURES

Figure 1.1: Illustration of the implicit even/odd extensions of DCT input data, for N=11 data points (rectangular region), for the four most common types of DCT 4

Figure 1.2: Diagram showing the MP3 Encoder after attaching an MDCT encoder. 5

Figure 1.3: Basic MP3 Audio encoder diagram illustrating how the different components are connected. 6

Figure 2.1. Block diagram showing the two main parts to the PEAQ algorithm 15

Figure 3.1: The coding flow diagram. 17

Figure 3.2. Generic perceptual audio coder. 19

Figure 3.4. MP3 Encoder Structure showing specific activities performed during audio encoding of the audio encoding chain, which provides a generalized overview of the flow of the coding involved in the development of the encoder in this project. 23

Figure 3.5: The Four parts of an MP3 frame 35

Figure 4.1: The MDCT project Front panel. 45

Figure 4.2: The recording interface 46

Figure 4.3: File compression summary window. 48

Figure 4.4: Audio1_32.wav waveform. 56

Figure 4.5: Audio1_32.mp3 waveform. 56

Figure 4.6: Area graph showing combinations of the waveforms in Figure 4.4 and Figure 4.5. 57

LIST OF LISTINGS

Listing 3.1: Part of the code for capturing the PCM signals..... 21

Listing 3.2: Setting the coefficients for the MDCT encoding. 25

Listing 3.3: The MDCT Filtering..... 26

Listing 3.4: The MDCT sub band Filtering 26

Listing 3.5: The Quantization process 30

Listing 3.6: The Huffman table selection criteria. 31

Listing 3.7: The frame Header format coding. 36

Listing 3.8: The Side information coding..... 40

Listing 3.9: Main data coding..... 43

ABSTRACT

Digital signal compression in the recent past has become the mainstay of the basic technologies of multimedia due to the bandwidth and storage limitations inherent in networks and computers. The change from circuit switched to packet switched systems in the telecommunication infrastructure in the past few years, has reflected on the way Pulse Code Modulation (PCM) signal which in essence is a raw audio signal, is carried in present systems. The purpose of this study was to design an application that utilizes type IV Discrete Cosine Transform (DCT IV), also known as the Modified Discrete Cosine Transform (MDCT) techniques to compress Sound of compact disk (CD) quality digital audio that has been encoded using the Motion Picture Experts Group (MPEG)-1 Audio Layer III (MP3) standard. Visual Basic.NET programming language has been used to achieve this. Although the compression technique used in this research project was a lossy one, the perceptual analysis showed no significant deviation between the raw uncompressed pulse code modulation (PCM) samples and the filtered MDCT output which uses psychoacoustic principles and MPEG psychoacoustic signal analysis to describe the hybrid filter bank design issues and algorithms. The results obtained from capturing raw audio signal at a bit rate of 128 kbps through three sampling rates namely 32 KHz, 41 KHz and 48 KHz were reviewed. The results showed a major improvement on the compression rate as a result of the MDCT application based on the MP3 compression standard. A significant data reduction of upto 90 percent of the original audio file was achieved when the MDCT coding was applied to raw uncompressed audio files based on the widely used MPEG-1 Audio Layer III (MP3) compression standard. In testing the quality of the compressed audio, subjective tests and the performance of Perceptual Evaluation of Audio Quality (PEAQ) software for objective audio quality measures for the prediction of the perceived quality of compressed files were investigated, and both the objective and subjective analysis showed high correlations.

CHAPTER ONE

INTRODUCTION

1.1 Background to the study

Data compression can be classified as either lossless or lossy (Brandenburg and Stoll, 2009). In a lossless compression, an exact copy of the original signal after decompression is produced while in the lossy compression, the original signal can no longer be reproduced. Lossless compression can successfully be applied in situations where there is plenty of data redundancy. It is easier to apply techniques like run-length suppression or Lempel–Ziv (LZ) compression. An example of a compression which maintains its original file is the ZIP file format. Lossless data compression is effective on several types of files (Jacaba, 2001). Lossless compressing of audio through this method is not as effective due to the fact that the information therein is less redundant (William, 1998). For this reason, we then prefer the lossy or perceptually lossless compression (Jacaba, 2001). Of the existing lossy encoding techniques, the Motion Picture Experts Group (MPEG)-1 Audio Layer III (MP3) standard, stands out as the most efficient and widely applied. The amount of data required to represent the audio signal recorded as PCM is tremendously reduced while its perceptual quality is still a good reproduction of the original uncompressed audio. An MP3 file with a setting of 128 kbit/s normally results in a file which is about 1/11 the size of the original audio file. Higher or lower bit rates can be used for MP3 files, with higher or lower output quality (Chiariglione and Leonardo, 2009).

The technology of MPEG is always developing. In the field of communication where speech compression is gaining ground, the application of this technology for instance, is

in areas like satellite communications. Examples are DBS TV's, TATA SKY, dish television network. They use MPEG-4, MPEG-1, MPEG-2 and MPEG-DASH. The compression of Speech has a streaming method so dynamic and adaptive over HTTP (DASH) technology. The technology is also being widely applied in cellular technology which includes mobile telephony as well as Voice over IP (Ramírez, 2008).

1.2 Theoretical Framework

The MPEG 1 standard was generally used with techniques and methods for compressing video and audio. MP3 is part of the MPEG 1 audio standard. Specifically, MP3 is "Layer 3" of MPEG 1. There are three 'layers' defined for MPEG 1 audio, with each layer being more complex than the previous and with each providing improved results. The layers are backwards-compatible, and therefore any software or hardware which can play (or 'decode') Layer 3 audio should consequently be able to decode both Layers 1 and 2 (Paul, 2000).

MP3 employs the "Discrete Cosine Transformation" (DCT) mathematical algorithm to compress signals. It takes a finite number of data points in sequence and in terms of a sum of cosine functions at different frequencies. Cosine functions use fewer functions than sine functions and for that reason are generally used to approximate a typical signal. For differential equations as well, the cosines express boundary conditions of choice (Narasimha and Peterson, 1978).

A DCT in particular is a Fourier-related transform which is similar to the Discrete Fourier transform (DFT), the only difference being its use of real numbers only. DCTs are twice the length of DFTs that operate on real data with even symmetry, where in some variants of these transforms, the input and/or output data are shifted by half a

sample. According to Literature, the DCT hase eight types of transforms, four of which are commonly used (Wen-Hsiung *et al.*, 1977).

The type-II DCT transform is the most common variant, and is often simply reffered to as “the DCT”. The inverse, which is the type-III DCT, is also called “the inverse DCT” or simply “the IDCT” correspondingly. The Sine counterparts are the discrete sine transforms (DST), which are equivalent tos DFTs which have both real and odd functions, and the type IV DCT also known as the modified discrete cosine transforms (MDCT), which is a transform based on a DCT of overlapped data (Johnson, 1987).

The MDCT, which is a lapped (Overlapping basically helps in avoiding distortions stemming from frame boundaries), is a Fourier transform which is based on type-IV discrete cosine transform (DCT-IV). It uses half as many outputs as inputs contrary to the same number as in other transforms, giving it a high compaction of energy. This is the main characteristic of MDCT that made it quite justifiable to apply as part of the lossy compression in our coder. The MDCT is a linear function F: $\mathbf{R}^{2N} \rightarrow \mathbf{R}^{N}$ where **R** denotes the set of real numbers. The 2N real numbers $x_0, ..., x_{2N-1}$ are actually the number of samples that were taken from the input signal (basically the MP3 outputs), and which were then transformed into the N real numbers $X_0, ..., X_{N-1}$ (the outputs of the MDCT transformation) according to the formula (Johnson *et al.*, 1987):

$$X_k = \sum_{n=0}^{2N-1} x_n \cos\left[\frac{\pi}{N}\left(n + \frac{1}{2} + \frac{N}{2}\right)\left(k + \frac{1}{2}\right)\right] \qquad 1.1$$

where k is any of the 1 to N outputs of the MDCT.

The different boundary conditions shown in figure 1.1 for DCT type I-IV affect the specific application of the transform, and it leads consequently to very unique and useful properties for the various DCT variants. For the MDCT, the boundary conditions are strongly involved in the MDCT's important property of time-domain aliasing

cancellation. The boundary conditions in a subtle way are the reason behind for the high energy compaction associated with the DCTs, and its usefulness is in image and audio compression. This is because the boundary behavior affects how much convergence there is in any Fourier-like series.

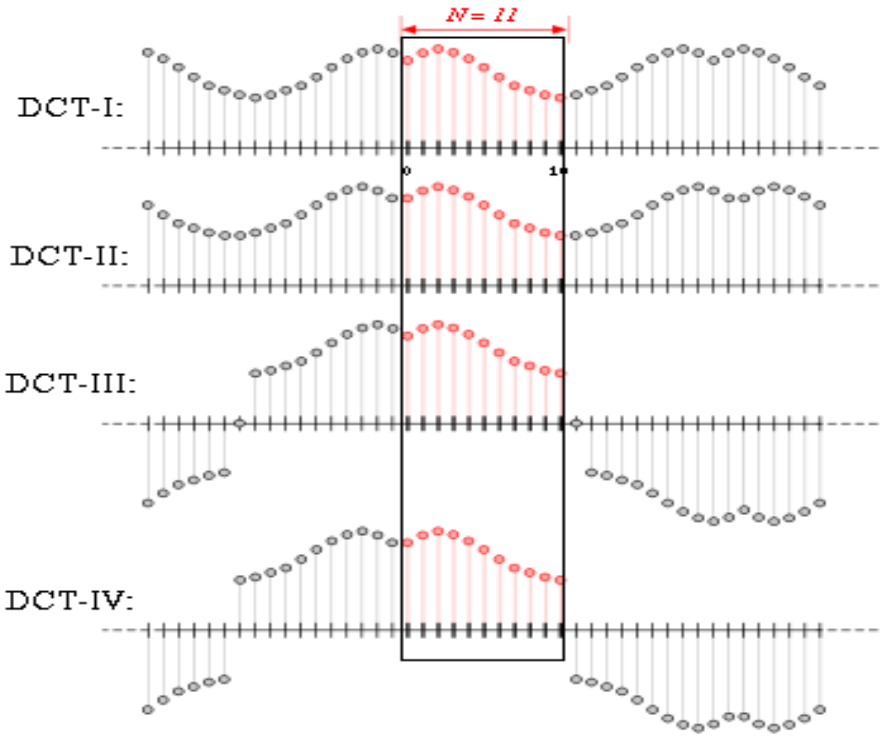

Figure 1.1: Illustration of the even/odd extensions of DCT data, for N=11 data points (rectangular region), for the four common types of DCT (types I-IV)(Rao and Yip, 1990) *http://en.wikipedia.org/wiki/Discrete_cosine_transform).*

The MDCT basically takes samples of the outputs of the MP3 encoder and filters them to produce a more compressed signal with a better quality (less artifacts) using a window function given by equation 1.2.

$$w_n = \left[\frac{\pi}{2n}\left(n + \frac{1}{2}\right)\right] \qquad 1.2$$

w_n (n = 0, ..., 2N-1) is the multiplicative function with X_n in the MDCT formula shown in equation 1.1. Its main function is help in avoiding discontinuities at the point where n

= 0 and 2N boundaries by ensuring that the function smoothly approaches zero at these points.

Recording equipment such as microphones and guitars, produce a wide range of audio resolutions most of which the human ear cannot perceive. When recorded in high fidelity, sound stores a lot of audio data that are inaudible, and by eliminating them, the final resulting audio sound will be indistinguishable from the originaly recorded. The MP3 format by this concept can attain compressions of ratios of up to 1:24. However, MPEG-1 standard for audio compression does not include a strict specification for an MP3 encoding (Davis, 1995). Any developer or researcher keen on implementing the standard is expected to devise own unique algorithms for removing portions of the information in the PCM audio. This consequently has led to many different MP3 encoders, each producing files of varying or different quality. This project did attach a Modified Discrete Cosine Transform (MDCT) encoding coupled with Huffman encoding to produce a lossy compression with better quality than pure MP3. The Huffman encoding was applied to reduce any redundancies that might exist within the compressed outputs. The flow diagram is as shown in figure 1.2.

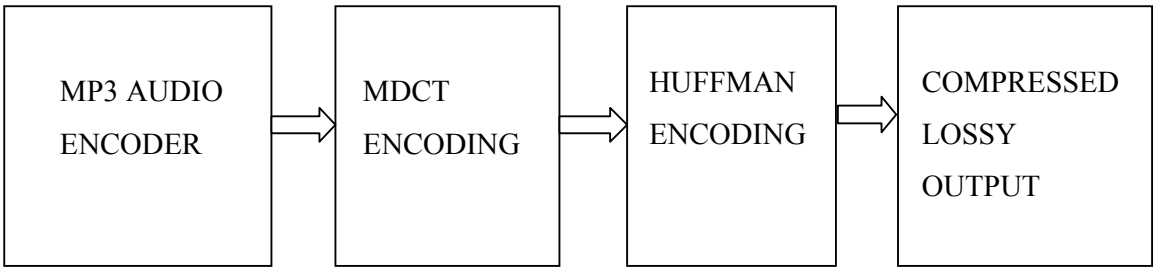

Figure 1.2: Diagram showing the MP3 Encoder after attaching an MDCT encoder.

The four main components of MP3, namely filterbank, psychoacoustics, quantization and bitstream formatting are shown in figure 1.3.

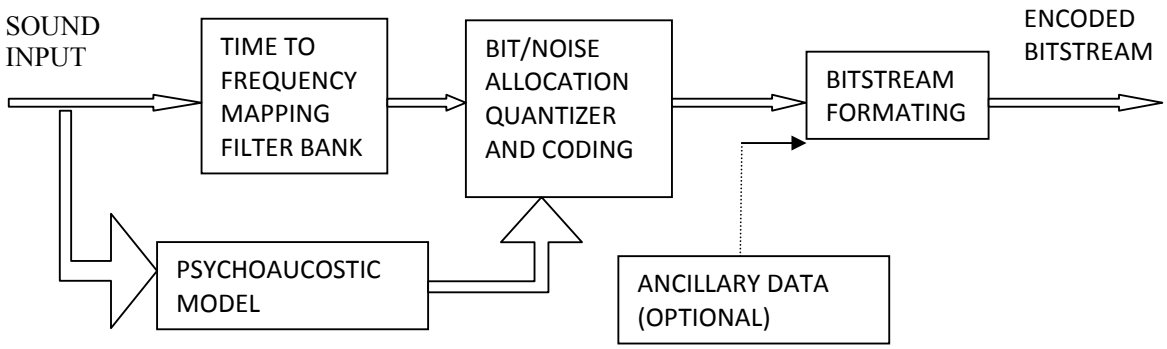

Figure 1.3: Basic MP3 Audio encoder diagram illustrating how the different components are connected.

1.3 Statement of the research problem

MP3 being one of the commonly used methods for compressing audio by removing irrelevant data from some recording, employs mathematical algorithms like the "Fast Fourier Transform" (FFT) or "Discrete Cosine Transform" (DCT) to achieve this (Peterson, 1978). Depending on the choice of algorithm chosen, the MP3 standard gives the freedom of how the irrelevant data in an input audio signal can be removed (Brandenburg and Stoll, 1987). The standard simply gives an outline of the necessary techniques and specifies the format for encoding the audio. MP3 is ambiguous and poorly written as it gives no details of implementation in some places. No emphasis is placed on the efficiency of computation. A new MP3 decoder is therefore a greater task than one would otherwise expect. Furthermore, the implementation of the MP3 technique during signal compression produces outputs accompanied by signal distortions (Schuller, 1998). In this study, a Modified discrete Cosine transform compression technique for sound optimization (MDCT) filter bank that adaptively switches between 128 and 1024 bands is implemented using Visual Basic.NET. The MDCT was used to double the number of frequency bands available for coding signals and which expectedly contributed to increased coding gain of this method.

1.4 Objectives

1.4.1 General Objective

The general objective of this study was to design an application that utilizes the Modified Discrete Cosine Transform (MDCT) techniques to compress audio files based on the MP3 standard.

1.4.2 Specific Objectives

(i). To create Visual Basic software components which would divide raw audio input into small Samples, i.e., the signal was to be made to pass into a time-frequency mapping filter bank which would divide it into sub bands.

(ii). To apply Psychoacoustic modeling to the sound samples. The concepts of masking and thresholds were to be utilized to discard data that was inaudible.

(iii). To quantize the output samples and then perform an MDCT filter on the output.

(iv). To apply both Subjective and Objective evaluation techniques to establish whether the MDCT encoding compromises on output signal quality.

1.5 Rationale/Justification of the Study

The polyphase filter component used in MP3 is obtained from an audio coder called Masking Pattern Adapted Universal Subband Integrated Coding and Multiplexing (MUSICAM) developed in earlier works (Princen *et al.*, 1987). It is a lowpass prototype filter which is cosine modulated and which has uniform bandwidth alongside an M-channel bandpass filter and has been named psuedo Quadrature Mirror Filter (QMF) (Princen *et al*., 1987). M in M-channel is equal to 32 in the case of MP3. The QMF output is generally succeptible to audible noise as it does not achieve high quality, also known as perfect reconstruction (PR). To circumvent this, the study endeavored to use MDCT as an addition, and which consequently formed a hybrid filterbank. MDCT restricts the sources of noise to the quantization stage (Princen *et al*., 1987).

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

The MPEG-1 Audio Layer III (MP3) format of signal compression used in encoding audio data uses a lossy compression. It is based on acoustics and psychoacoustics which considers the way the human ear interprates signals. This phenomenon is also known as the perceptive behavior of the human ear. MPEG is the official standardization by a committee formed under the International Electro technical Commission (IEC) and the International Standards Organization (ISO), for the sole purpose of developing international standards which guides the efficient encoding of video for motion picture and high quality and fidelity audio. In 1993, the MPEG committee published the MPEG standard in the form of the document ISO/IEC 11172. It therefore set out a common standard for the coding of video or moving pictures and the associated audio signals for digital transmission and storage in digital media at rates of up to about 1.5 Mbit/Second, and it was named the MPEG 1 standard (Paul, 2000).

Davis (2005) explains in the article "A Tutorial on MPEG/Audio Compression", that the human auditory system has some special properties which can be exploited in perceptual audio manipulation. The human ear has a dynamic range of frequencies from about 20 to 20000 Hz, and can hear sounds with varying intensity over several varying magnitudes. The human auditory system thus behaves like a very wide-ranged instrument, but which is not the case (Dobson *et al*., 1997), what the ear hears depends on the audio environment. In an environment where there is a strong white noise, much of the weaker audio sounds get masked, and may not be heard at all. Studies show that some of these masking characteristics of the human ear are due to its physical appearance, and some

are as a reason of how the processing of signals in the brain takes place. It therefore means that these masked sounds can effectively be filtered off without affecting the quality of the final audio signal, thus justifying why MP3 standard, which is a lossy compression technique was considered in this study as a basis for the MDCT compression.

Audio Compact Disks (CDs) use the popular waveform (WAV) format, which is uncompressed (Davis, 1995). The WAV format falls in many categories one of which is the Pulse Code Modulation (PCM) which is the commonly accepted MP3 input during enoding. The WAV file size depends on the rate of which it is sampled. As an example, an 8-bit mono WAV sampled at 22,050 Hertz (Hz) would take 22,050 bytes per second. A 16-bit stereo WAV file which has been sampled at the rate of 44.1 kilohertz (kHz) would take 176,400 bytes per second (this has been calculated as 44,100 per second * 2 bytes * 2 channels). Calculations of a one minute of an audio file of CD-quality takes a storage space of 10 MB. It is therefore necessary to find methods and algorithms for reducing huge requirements in data storage of such files. MP3standadrd is one of the methods mostly used for this purpose (Jacaba, 2001).

2.2 Lossless and Lossy coders

High quality CD or audio Data require large amounts of memory storage space. A CD stream has 44100 samples per channel. This is equivalent to 1.4 Mbits/second when recorded in stereo (Martin, 1992). At this rate, the audio file contains a lot of redundant data, which can be removed in a lossless coder. A lossless coder does not change the data, but the original audio is completely recovered after decompression. Lossless audio coders can get the bit-rate down to about half the original rate, or 700 kbit/s. A lossy coder which exploits redundancies can be made much more efficient without perceptual loss to the original signal.

2.3 Existing Coders and Standards

Several implementations of perceptual coders exist in commercial use. Those considered to be top-of-the-line include Sonys ATRAC, which is used in the MiniDisc system. MPEG-1 Layer I, II and III standards are the other well known examples that implement perceptual coding (Brandenburg and Stoll, 1987). MPEG-2 Advanced Audio Coding (AAC) and Dolby's AC-3 (Bosi et al., 1997). The differences in the compression technique between the many coders are in the way they model the human ear and the type of subband coding used. The general problem in these coders is how to handle certain signal artifacts, which are noticeable distortion of audio media caused by the application of lossy approach in the compression of data. Currently, the MP3 standard is the format commonly used for audio streaming or storage in consumer products. It is also the de facto standard for digital compression of audio as well as the transfer and playback on most digital audio players (Chiariglione and Leonardo, 2009).

2.3.1 Subband Coders

Subband splitting or filtering of a signal is a method used over and over to refine bands in a filter tree (Princen *et al*, 1997). The splitting of a 20000Hz signal for instance, would be split as:

$$0 – 20000 \text{ Hz} \rightarrow 0 - 10000 \text{ Hz}, 10000 - 20000 \text{ Hz}$$

$$0 - 10000 \text{ Hz} \rightarrow 0 - 5000 \text{ Hz}, 5000 \text{ Hz} - 10000 \text{ Hz}$$

The coder in this way can divide the input audio data into several equal bands, where each of which a quantizer, adapted to the masking threshold, is finally applied.

The four filters of the filters bank are, $F_0(w)$, $F_1(w)$, $G_0(w)$, and $G_1(w)$, where the F's are for splitting the input data in both high and low frequency band. The split bands are sub-

sampled using a factor of 2. To reconstruct the signal, the two bands are again up-sampled and filtered using the *G*'s. The output expression for $\hat{X}(\omega)$ is as shown in equation 2.1;

$$\hat{X}_{(w)} = \frac{1}{2}\left(F_{0(w)}G_{0(w)} + F_{1(w)}G_{1(w)}\right)X_{(w)} + \frac{1}{2}\left(F_{0(w+\pi)}G_{0(w)} + F_{1(w+\pi)}G_{1(w)}\right)X_{(w+\pi)} \qquad 2.1$$

Aliasing cancellation for the filters $G_0(w)$, and $G_1(w)$, is obtained by the requirements of equations 2.2 and 2.3 (Princen *et al*, 1997).

$$G_{0(w)} = F_{1(w+\pi)} \qquad 2.2$$

$$-G_{1(w)} = F_{0(w+\pi)} \qquad 2.3$$

Optimally,

$$\frac{1}{2}\left(F_{0(w)}G_{0(w)} + F_{1(w)}G_{1(w)}\right) = 1, w \in [0..\pi] \qquad 2.4$$

With these equations, a good reconstruction is achieved. In most coders, the filters used are Quadrature Mirror Filters (QMF). They manage to perform distortion or aliasing eradication by choosing F_1 such that it is the mirror image of F_0 (around π/2):

$$F_{1(w)} = F_{0(w+\pi)} = -G_{1(w)} = G_{0(w+\pi)} \qquad 2.5$$

For a good reconstruction of audio signals, the QMF filters are generally required to have a transition of pass-to-stop-band which is steep (Princen *et al*, 1997).

2.3.2 MPEG-1, Layer I and II

The ISO MPEG-1 coding standard has layers I, II and III all of which differ in coding and psychoacoustic models (Davis, 2005).

Both Layers I and II use a QMF filter bank of order 511, and which has a 96 dB rejection of side-lobes and also has a steep transition of pass-to-stop-band. The filter bank splits the the audio input into 32 sub-bands.

MPEG layer III uses the psychoacoustic model which consists of a 512 point Fast Fourier Transform (FFT), from which a masking threshold is calculated using equation 3.2 shown in chapter 3 section 3.3.2. Here, the maximum values of the spectrum are considered as components which influence the tonal portion of the signal and are therefore treated differently from the noise components. To produce a masking threshold, both the noise and the tonal masking thresholds are added. A linear quantizer is used together with the masking threshold to achieve the required performance in the rate-distortion in each sub-band. Eventually each sub-band will then have the same noise-to-mask-ratio and the bits add up to the appropriate bit-rate. The quantized sub-band is fixed-length encoded rather than running length encoded.

2.3.3 Wavelet Coders

The transformation of a wavelet can be seen as a breaking down of sub-bands. The difference thereof nevertheless, is that the generated sub-band tree in wavelet decomposition is often unbalanced, which implies that the splitting of frequency is not uniform. This way, a filter bank similar to the critical bands can easily be formed (Dobson *et al.,* 1997).

2.4 Modified Discrete Cosine Transform (MDCT)

It is also referred to as MPEG-2 AAC and is designed to take the place of the MP3 compression technique. In MDCT, the input audio data is initially split using a Polyphase Quadrature Filter (PQF) into four uniform subbands (Bosi *et al.*, 1997). An individual gain is then transmitted as side information for each subband. The subband data controlled by the gain is then transformed using an MDCT whose frame length is 256 (for transient conditions the length is 32). The sine window of equation 1.2 in chapter one is used to smothen the boundary artifacts for the MDCT. The windowing function has different spectral characteristics, which are suitable for a variety of signals. For conditions which are transient in nature, a shorter window is used for improved resolution of time. The coefficients of the MDCT are computed from two preceding frames, using a separate Least Mean Square (LMS) predictor adapted for every frequency band. This method when applied to stationary signals improves coding efficiency (Bosi *et al.,* 1997).

The MDCT transformation is understood on the basis of the knowledge of Time Domain Aliasing Cancellation (TDAC) and is also a linear orthogonal transform which is lapped for better energy compaction (Princen and Bradley, 1986; Princen *et al.,* 1987).

In MDCT, though there is a 50% overlapping, a sequence of data processed after MDCT and the samples before transformation have the same number of coefficients. It can therefore be deduced that a block of Inverse modified transform (IMDCT) audio data consequently has no one to one correspondence to the original MDCT block on which the transformation was performed. Adding subsequent blocks of inverse transformed data will cancel out errors introduced by the transformation while still using 50% overlap. This method of error cancellation is also called time domain aliasing cancellation (TDAC). The MDCT overlap feature makes it very useful for quantization.

It is very effective in removing blocking artifacts that would have otherwise been easy to detect between the transform blocks. The definition of MDCT is summarized in equation 2.6, (Sporer *et al,* 1997).

$$X_{(m)} = \sum_{k=0}^{n-1} f(k)x(k)\cos\left(\frac{\pi}{2n}\left(2k+1+\frac{n}{2}\right)(2m+1)\right), for\ m = 0..\frac{n}{2}-1 \qquad 2.6$$

The corresponding IMDCT is shown in equation 2.7.

$$Y_{(p)} = f(p)\frac{4}{n}\sum_{m=0}^{\frac{n}{2}-1} X(m)\cos\left(\frac{\pi}{2n}\left(2p+1+\frac{n}{2}\right)(2m+1)\right), for\ p = 0..n-1 \qquad 2.7$$

where $f(x)$, is the window function explained in equation. 1.2.

In this coder, the MDCT transformation was achieved using frames whose lengths are 512, producing 256 samples which were then used consecutive blocks.

2.5 Perceptual Evaluation of Audio Quality (PEAQ)

The International Telecommunications Union (ITU) proposes a standard that can be used to asses the quality of audio files after has undergone some transformation. This standard which as of the writing of this thesis was the only method used for assessing audio quality is known as the Perceptual Ealuation of Audio Quality (PEAQ). It is mostly used while developing codecs and networks and also for testing multimedia devices for audio playback quality. ITU-RBS.1387 standardized the PEAQ during 1998–2001 (Campbell *et al*, 2009). According to Dermot Campbell *et al*, PEAQ besides the aforementioned application can also be used to distinguish between different audio devices in terms of output quality. It has initially been a traditional practice to use subjective means to assess audio quality. It involves human listening tests which from experience gathered by audio quality testers are expensive and take a long time. It is in the light of this that computer

based algorithms which are designed to facilitate objective assesment of the quality of audio devices, computer networks and digital systems. The one such algorithm is the PEAQ which essentially models the behavior of the human ear. In other words it is designed based on the psychoacoustic principles of the human auditory system. It ia designed to maintain an acceptable level of audio quality despite a reduction in the number of bits of an audio signal. PEAQ consists of two parts namely the psychoacoustic model and the cognitive model. The overview is shown in figure 2.1. In figure 2.1, the original undistorted signal is used as the reference. The degraded input represents the distorted output test signal that needs assessment. The final scores are given as values in the range 0 to -4.

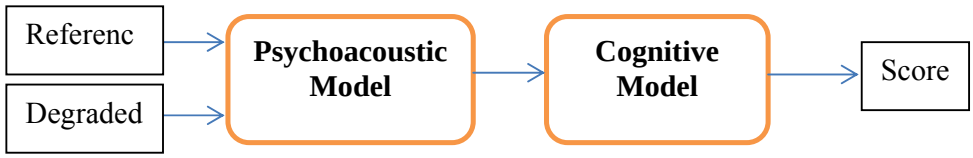

Figure 2.1: Block diagram showing the two main parts to the PEAQ algorithm (Campbell, 2009).

There are two vesrions of PEAQ: the basic version and the advanced version. The former is intended for those applications requiring high processing speed, and the latter is for applications requiring highest accuracy (Vanam, 2005).

The PEAQ basic version utilizes an FFT-based psychoustic model. The advanced version on the other hand uses the FFT-based rmodel together with a filterbank-based model of the human ear. The concept of sound masking is implemented with the FFT-based ear model, and the concept of internal representations comparisons is implemented using the filterbank-based ear model. There are eleven model output variables (MOV) jn

the basic version. They are obtained from the model of the human ear for the purpose of measuring pitch and tone. There are five MOVs for the advanced version. They are derived from the filter-bank, pitch and other parameters associated with audio quality. In addition, the FFT-based MOVs include a noise-to-mask ratio in the error signal. In the calculation of the MOVs, spectral values over frequency bands are averaged and also a temporal averaging over temporal frames is computed. The signals undergo frame selection during the computation of the MOVs. It is either from the basic version or the advanced version that the MOVs are extracted. Once the extraction is complete, the MOVs are mapped to a value or an indeces corresponding to objective difference grades (ODG). This is achieved by neural network software which applies a multi-layer with either 3 (for the basic version) or 5 (for the advanced version) units. The training of the neural network software is done using all available data (Junyong *et al*, 2010).

CHAPTER THREE

RESEARCH METHODOLOGY

3.1 Coding:

The study was code intensive. Part of the Code was purely for capturing raw sound signal from a microphone or sound card, in form of PCM for storage before being compressed further using MP3 and finally MDCT. The code modulation followed the steps shown in figure 3.1.

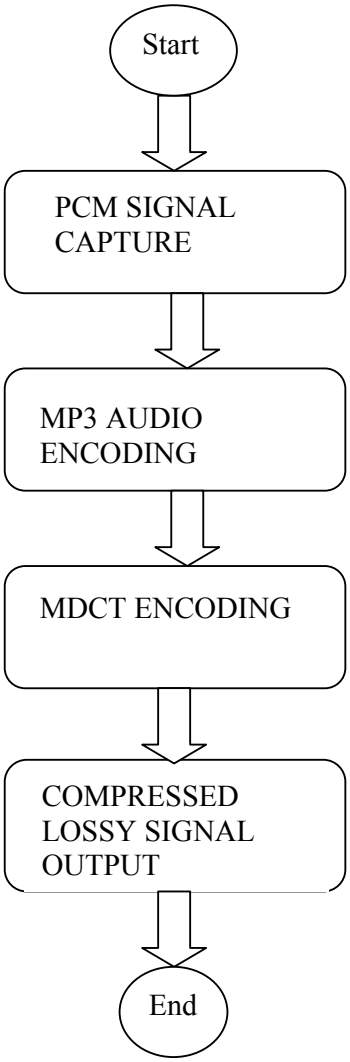

Figure 3.1: The coding flow diagram.

3.2 Algorithm:

1. Read an uncompressed audio format file, PCM, which is usually stored in a waveform (WAV) file on Windows or in an audio interchange file format (AIFF) file on Mac OS. Windows waveform (WAV) was used in this study.
2. Use Convolution filters to divide the input signal into subbands containing 32 critical bands.
3. Compute the masking values for every band due to the bands in its neighbourhood using the results from step 2.
4. No encoding for a band whose power is below the masking threshold.
5. Determine how many bits are required to represent the coefficient with values that will ensure noise introduced by quantization is no more than the effects of masking.
6. Create an MDCT Object to format the output of previous steps.
7. Run MDCT object on all blocks read from Step 6 above.
8. Run the quantization function on all blocks.
9. Finally save the compressed signal in file.
10. Play the audio signal.

3.3 Algorithm description and Analysis

The encoder Algorithm development, had several steps as outlined in section 3.2, and generally followed the flow chart depicted in figure 3.1. In this section, the steps shown in figure 3.1 are analyzed in detail. The audio data lossy compression technique described here achieves coding gain by exploiting statistical redundancies and perceptual irrelevancies. The general outline is given in figure 3.2 (Painter, 2000).

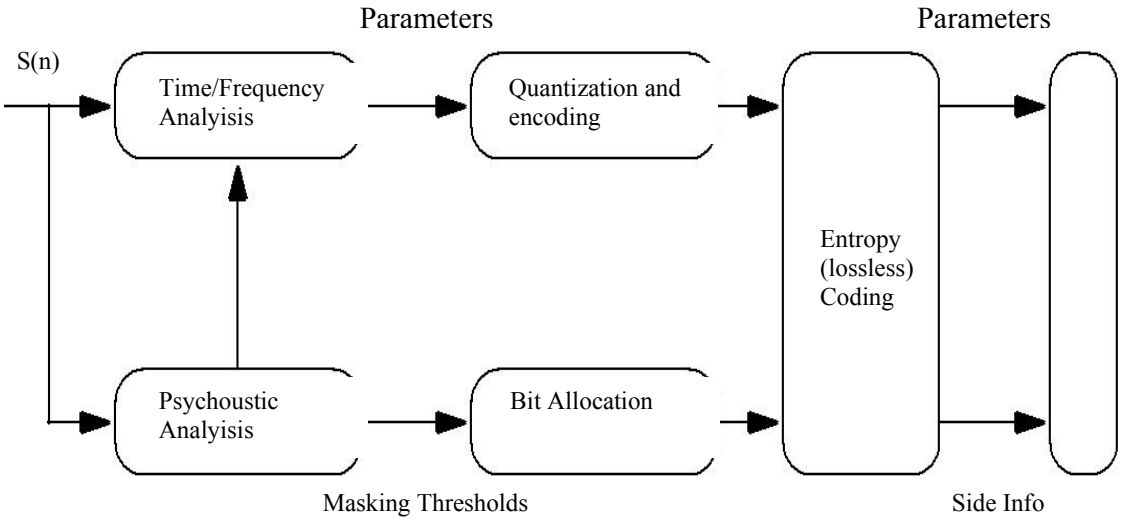

Figure 3.2. Generic perceptual audio coder (Painter and Spanias, 2000).

Painter and Spanias (2000) describe that the coder in figure 3.2, creates segments of quasistationary frames from the input audio s (n) which range from 2 to 50 ms in a given time. Next the analysis block for the time-frequency mapping computes an estimation of the temporal and spectral components found in each given frame. A mapping of these components to the properties used for analysis of the behavior of the human auditory system and then finally the time to frequency mapping parameters that will be used for quantization and eventually encoding are extracted.

The block responsible for psychoacoustics which is also called the psychoacoustic block permits for the quantization and consequently the encoding block to take advantage of information that cannot be perceived in the time to frequency set of parameter. The remainders of the redundancies are ususlly removed through the process of lossless entropy coding techniques.

3.3.1 PCM Signal capture

The Audio signal represented by sine waves of varying frequencies, is initially in an Analog format, meaning that it is a continuous signal. The human ear has the ability to percieve frequency nominally in the range of 20 to 20,000 cycles/second (Hertz or Hz). To Store sound in a digital format, it is required that the sound wave which is continuous be converted to a group of samples that can then be represented by a train of zeros and ones.

Conversion of the signals from Analog to digital formats was accomplished by sampling the analog signal using a microphone, after which the signals were converted to equivalent voltage levels. The voltage or signal levels were converted into digital bits by use of a coder-decoder or codec to a numeric value using a pulse code modulation (PCM) method. The code in listing 3.1 shows the section of the code that was used to capture the raw uncompressed audio signal.

PCM only provides values which are close to the actual audio signal or approximation of the original analog signal, as shown in figure 3.3. If for example the analog sound level is measured at a 4.88 level, it would then be converted to a value of five in pulse code modulation. This difference between these values is called the quantization error. The audio samples in this project were captured using Sixteen bits. The 16-bit values use the numbers 0 to 65,535 to represent samples (Sporer,1997).

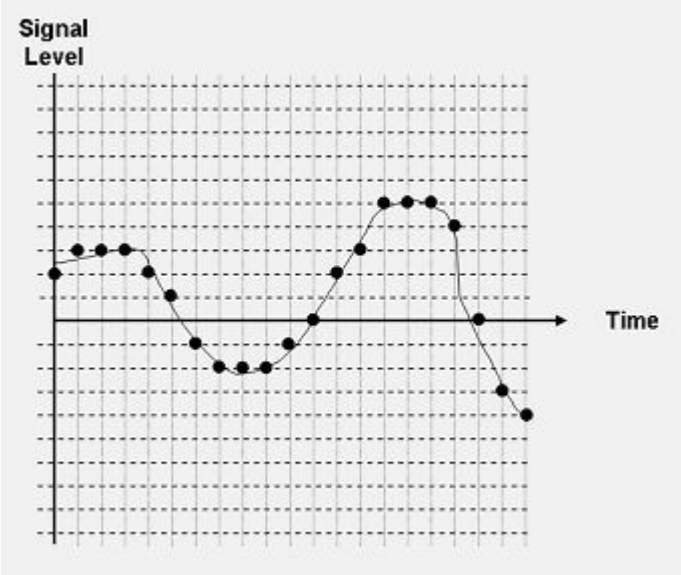

Figure 3.3: Simple Pulse Code Modulation

For a good sampling, analog signals need to be recorded at a sampling frequency which is twice the highest frequency value of the input signal according to Nyquist sampling theory; this is so that the original signal can be reproduced to a near resemblance from the samples. Applications of Music audio always assume the full wave spectrum of the human ear i.e from 20Hz to 20000Hz and generally use a 44.1 k (Painter, 2000).

Listing 3.1: Part of the code for capturing the PCM signals

```
Private Sub StartOrStopRecord(ByVal StartRecording As Boolean)
    ' Starts or stops the capture buffer from recording
    Writer = New BinaryWriter(WaveFile)
    If Recording = StartRecording Then
      ' Create a capture buffer
      ' buffer to start recording
      CreateCaptureBuffer()
```

```
            Recording = True
            applicationBuffer.Start(True)
        Else
            ' Stop the buffer
            applicationBuffer.[Stop]()
            RecordCapturedData()
            Writer.Seek(4, SeekOrigin.Begin)
            ' Seek to the length descriptor of the RIFF file.
            Writer.Write(CInt(SampleCount + 36))
            Writer.Seek(40, SeekOrigin.Begin)
            Writer.Write(SampleCount)
            Writer.Close()
            Writer = Nothing
            WaveFile = Nothing
            'Recording = False
            Capturing = False
            'NotificationEvent.Set()
        End If
    End Sub
```

Before the audio file is compressed, its bit rate can be calculated from its sampling rate which is a 44.1 kHz for most practicle cases and its code modulation resolution which is 16 bits together with two channels are used to calculate the bit rate and found to be 1,411,200 bits per second. This means that a one-minute uncompressed audio file would

occupy nearly 10.6 MB (1,411,200 bits * 60 seconds / 8 bits = 10,584,000 bytes). In essence, audio files are made smaller by compressing them using a several available techniques. A quick method for doing so is to reduce the number of channels from two to one or to reduce the sampling rate to a lower value than 44.1 KHz, in some cases to as low as 11 kHz. Any of the chosen solutions will in some way reduce the quality of the sound. Here we used the Fast Fourier Transform (FFT) block to transform granules of 576 samples to the frequency domain using Fourier Transform.

3.3.2 MP3 audio encoding

The MP3 encoder has the structure of figure 3.4 (Pan, 1995):

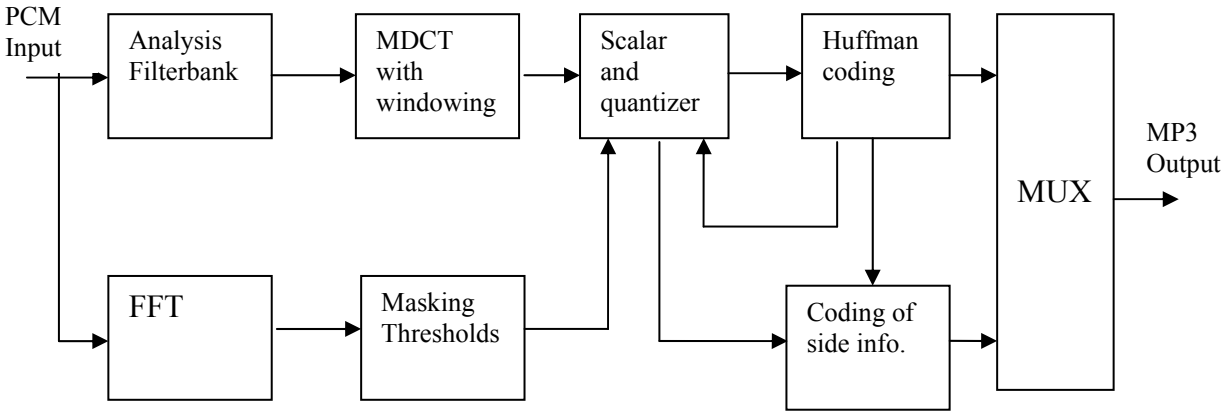

Figure 3.4: MP3 Encoder Structure showing specific activities performed during audio encoding of the audio encoding chain, which provides a generalized overview of the flow of the coding involved in the development of the encoder in this project.

The PCM input data, after been sampled was stored in disk as uncompressed file with a .wav extension. The file was then split into frames of 1152 samples each. Each of the frames was further divided into two granules of 576 samples per frame. The frames were then processed by the FFT block for correlation before finally passing through the analysis filterbank.

The main function of the FFT algorithm was to reduce the number of computations for N points from $2N^2$ to N log N, where the log is in base-2 logarithm. Fourier transforms that use base-4 and base-8 are normally optimized code, which can be 20-30% faster than base-2 FFTs. The FFT function itself is given by equation 3.1 (Painter, 2000).

$$\sum_{n=0}^{N-1} a_n e^{-2\pi i n k/N} = \sum_{n=0}^{N/2-1} a_{2n} e^{-2\pi i (2n) k/N} + \sum_{n=0}^{N/2-1} a_{2n+1} e^{-2\pi i (2n+1) k/N} \qquad 3.1$$

There first term on the right hand side of equation 3.1 above represents the even coefficients while the second term represents the Odd coefficients of the samples within the signal.

The results obtained from applying the FFT, which are now in the frequency domain, were used together with a psychoacoustic model to determine the masking thresholds for all frequencies integrated within the audio signal. The masking thresholds formula is given by equation 3.2 (Terhardt, 1979).

$$Threshhold = 3.64\,X\left(\frac{f}{1000}\right)^{-0.8} - 6.5\,X\,e^{-0.6X\left(\frac{f}{1000}-3.3\right)^2} + 10^{-3}\,X\left(\frac{f}{1000}\right)^4 \qquad 3.2$$

where *f* represents frequency (Hz) and dB is the unit of threshold.

3.3.3 MDCT

Once the 32 bandpass filters of equal width had been passed to the FFT analysis filterbank, their output were critically sampled by the type IV DCT or the Modified Discrete Cosine Transform (MDCT). Thus for each granule containing 576 samples there were 18 samples produced as output from each of the 32 bandpass filters, which gave a total of 576 subband samples.

Higher frequency resolution was achieved by the use of DCT and the control of the pre-echo (Painter, 2000). The application of MDCT to every time frame of the given subband samples splits the 32 subbands into 18 smaller and more refined subbands, and a granule with a total of 576 frequency lines is thus created.

To reduce the noise caused by the boundaries of the signal in the spatial domain before performing the MDCT, each subband signal had to be windowed with the sine function shown in equation 1.2. The MDCT is performed on blocks that are windowed and overlapped by 50%. Listing 3.2 shows how the coefficients of the MDCT are determined and listing 3.3 and 3.4 are the MDCT filtering and sub band filtering methods respectively.

Listing 3.2: Setting the coefficients for the MDCT encoding.

```
Private Sub mdct_initialise()
    Dim i, m, k As Integer 'N;
    Dim sq As Double
    ' prepare the aliasing reduction butterflies
    For i = 7 To 0 Step -1
       sq = Math.Sqrt(1.0 + Math.Pow(c(i), 2))
       ca(i) = c(i) / sq
       cs(i) = 1.0 / sq
    Next i
    For i = 35 To 0 Step -1
       win(i) = Math.Sin(Math.PI * 36 * (i + 0.5))
```

```
        Next i
        'N = 36;
        For m = 0 To 17
            For k = 0 To 35
                cos_l(m, k) = Math.Cos((Math.PI / (72)) * (2 * k + 19) * (2 * m + 1)) / (9)
            Next k
        Next m
    End Sub
```

Listing 3.3: The MDCT Filtering

```
    Private Sub mdct(ByVal SamplesIn() As Double, ByVal SamplesOut() As Double)
        Dim k, m As Integer
        For m = 18 To 0 Step -1
            SamplesOut(m) = win(35) * SamplesIn(35) * cos_l(m, 35)
            For k = 35 To 0 Step -1
                SamplesOut(m) += win(k) * SamplesIn(k) * cos_l(m, k)
            Next k
        Next m
    End Sub
```

Listing 3.4: The MDCT sub band Filtering

```
Private Sub mdct_sub(ByVal sb_sample(,,,) As Double, ByRef mdct_freq(,) As Double,
ByRef side_info() As side_info_t)
        ReDim sb_sample(2, 3, 18, SBLIMIT)
```

```
        ReDim mdct_freq(2, samp_per_frame2)
        Dim mdct_enc(2, 32, 18) As Double
        'mdct_enc(2, 32, 18) = mdct_freq(2, 32, 18)
        Dim ch, gr, band, k, j As Integer
        Dim cod_info() As gr_info
        Dim mdct_in(36) As Double
        Dim bu, bd As Double
        For gr = 0 To gr < 2
            For ch = config.wave.channels To 0 Step -1

                'cod_info = (gr_info*) &(side_info.gr(gr).ch(ch))
                ' Compensate for inversion in the analysis filter
                For band = 32 To 0 Step -1
                    For k = 18 To 0 Step -1
                        If ((band And 1) And (k And 1)) Then
                            sb_sample(ch, gr + 1, k, band) *= -1.0
                        End If
                    Next k
                Next band
' Perform imdct of 18 previous subband samples + 18 current subband samples
                For band = 32 To 0 Step -1
                    For k = 18 To 0 Step -1
                        mdct_in(k) = sb_sample(ch, gr, k, band)
```

```
                mdct_in(k + 18) = sb_sample(ch, gr + 1, k, band)
            Next k
            mdct(mdct_in, mdct_enc(gr, ch, band), 0)
        Next band
        ' Perform aliasing reduction butterfly
        For band = 31 To 0 Step -1
            For k = 8 To 0 Step -1
                bu = mdct_enc(gr, ch, band, 17 - k) * cs(k) + mdct_enc(gr, ch, band + 1,
k) * ca(k)
                bd = mdct_enc(gr, ch, band + 1, k) * cs(k) - mdct_enc(gr, ch, band, 17 - k)
* ca(k)
                mdct_enc(gr, ch, band, 17 - k) = bu
                mdct_enc(gr, ch, band + 1, k) = bd
            Next k
        Next band
    Next ch
Next gr
' Save latest granule's subband samples to be used in the next mdct call
For ch = config.wave.channels To 0 Step -1
    For j = 18 To 0 Step -1
        For band = 32 To 0 Step -1
            sb_sample(ch, 0, j, band) = sb_sample(ch, 2, j, band)
        Next band
```

```
        Next j
    Next ch
End Sub
```

Listing 3.4 shows the portion of the code used to create the overlapping effect of the MDCT whose inputs are 2N and whose outputs are N. It achieves this by dividing the inputs of the signal into four blocks of size N/2 each. If these inputs are shifted by N/2 then the end of the N DCT-IV inputs will create a "fold back" according to the signal edge conditions described in equation 3.3 and equation 3.4. Thus, the MDCT of 2N inputs is exactly the same as that of a DCT-IV of N inputs.

$$\cos[\pi/N\,(-n-1+1/2)(k+1/2)] = \cos[\pi/N\,(n+1/2)(k+1/2)] \qquad 3.3$$

$$\cos[\pi/N\,(2N-n-1+1/2)(k+1/2)\,] = -\cos[\pi/N\,(n+1/2)(k+1/2)] \qquad 3.4$$

3.3.4 Compressed signal output formatting

The number of bits required or needed for each critical band to represent the samples so that noise is minimized was determined by use of masking thresholds. Afterwards the psychoacoustic model was used to allocate bits.

The encoder iteratively and in an ordely manner, changes the quantizers as demonstrated in the code portion shown in listing 3.5. It compacts the energy of the different spectral values, then determines the number of bits needed for the Huffman encoding to code the file data, and estimate the generated noise.

Listing 3.5: The Quantization process

```
Private Sub quantize(ByVal xr() As Double, ByVal ix() As IntPtr, ByRef cod_info As
gr_info)
    Dim i As Integer
    Dim idx As Integer
    Dim steps As Double
    Dim dbl As Double
    Dim ln As Long
    steps = Math.Pow(2.0, (cod_info.quantizerStepSize) / 4)
    For i = samp_per_frame2 To 0 Step -1
      dbl = Math.Abs(xr(i)) / steps
      ln = CLng(dbl)
      If (ln < 10000) Then
        idx = int2idx(ln)
        If (dbl < idx2dbl(idx + 1)) Then
          ix(i) = idx
        Else
          ix(i) = idx + 1
        End If
      End If
    Next
  End Sub
```

The encoder amplifies values whose noise levels are too high within the scale factor bands, and then effectively reduce the quantizer step size for those values in the affected

bands. According to (Pan, 1995), the process stops depending on when any of the conditions below is met:

a) When the scale factor bands contain less distortions than the accepted value

b) The proceeding iteration causes excess amplification for any of the bands.

c) The proceeding iteration requires that all values of the scale factor bands be increased.

The iterations include the Huffman coding since it is needed for determining the number of bits needed for the encoding. Listing 3.6 shows the section of program code that helps choose the specific Huffman values to be used during the quantization process.

After the samples have been quantized, they are Huffman-coded and finally stored in the bitstream with the scale factors and side information.

Listing 3.6: The Huffman table selection criteria.

```
Function new_choose_table(ByVal ix() As Integer, ByVal begin As UInteger, ByVal
ends As UInteger) As Integer

'*************************************************************/
' Choose the Huffman table that will encode ix[begin..end] with   */
'the fewest bits.                                    */
    '*************************************************************/
    ix(samp_per_frame2)
    Dim i, max As Integer
    Dim choice(2) As Integer
    Dim sum(2) As Integer
```

```
max = ix_max(ix, begin, ends)
If (Not max) Then
   Return 0
End If
choice(0) = 0
choice(1) = 0
If (max < 15) Then
   ' try tables with no linbits */
   For i = 14 To 0 Step -1
      If ht(i).xlen > max Then
         choice(0) = i
         Exit For
      End If
      'End If
   Next i
   sum(0) = count_bit(ix, begin, ends, choice(0))
   Select Case choice(0)
      Case 2
         sum(1) = count_bit(ix, begin, ends, 3)
         If sum(1) <= sum(0) Then
            choice(0) = 3
         End If
```

```
Case 5
    sum(1) = count_bit(ix, begin, ends, 6)
    If sum(1) <= sum(0) Then
        choice(0) = 6
    End If
Case 7
    sum(1) = count_bit(ix, begin, ends, 8)
    If sum(1) <= sum(0) Then
        choice(0) = 8
        sum(0) = sum(1)
    End If
    sum(1) = count_bit(ix, begin, ends, 9)
    If sum(1) <= sum(0) Then
        choice(0) = 9
    End If
Case 10
    sum(1) = count_bit(ix, begin, ends, 11)
    If sum(1) <= sum(0) Then
        choice(0) = 11
        sum(0) = sum(1)
    End If
    sum(1) = count_bit(ix, begin, ends, 12)
    If sum(1) <= sum(0) Then
```

```
            choice(0) = 12
        End If
    Case 13
        sum(1) = count_bit(ix, begin, ends, 15)
        If sum(1) <= sum(0) Then
            choice(0) = 15
        End If
End Select
Else
    max -= 15
    For i = 15 To 23
        If ht(i).linmax >= max Then
            choice(0) = i
            Exit For 'break;
        End If
    Next
    For i = 24 To 31
        If ht(i).linmax >= max Then
            choice(1) = i
            Exit For
        End If
    Next
```

```
        sum(0) = count_bit(ix, begin, ends, choice(0))
        sum(1) = count_bit(ix, begin, ends, choice(1))
        If sum(1) < sum(0) Then
            choice(0) = choice(1)
        End If
    End If
    Return choice(0)
End Function
```

3.3.5 Frame Format

1152 mono or stereo spectral samples samples form part of the frame. These samples are each divided into two granules of 576 samples each as demonstrated in listing 3.4. A further division of each granule produces 32 subband blocks of 18 frequency lines each. The frequency spectrum ranges from 0 to $F_S/2$ (Sampling frequency divided by 2) Hz.). The frame consists of the header, side information, main data, and ancillary data:

Header	Side information	Main data	Ancillary data

Figure 3.5: The Four parts of an MP3 frame (Lagerström, 2001).

Listing 3.7 comprises the code that dequantizes processed samples and packaging the frame headers, which is later joined with the side information, main data and the Ancillary data to form the frame.

Listing 3.7: The frame Header format coding.

```
Private sub Frame_Header(byval is(SBLIMIT,SSLIMIT) as long , byval
xr(SBLIMIT,SSLIMIT) as double , _
                    byref scalefac as III_scalefac_t, byref gr_info As gr_info_s, _
                    byval ch As Integer , byref fr_ps As frame_params *)
    Dim ss, sb As Integer
    Dim cb As Integer = 0
    sfreq = fr_ps.header.sampling_frequency
    Dim stereo As Integer = fr_ps.stereo
    Dim next_cb_boundary, cb_begin, cb_width, sign As Integer
    ' choose correct scalefactor band per block type, initalize boundary
    If gr_info.window_switching_flag And gr_info.block_type = 2 Then
      If gr_info.mixed_block_flag Then
        next_cb_boundary = sfBandIndex(sfreq).l(1)  'LONG blocks: 0,1,3
      Elseif
        next_cb_boundary = sfBandIndex(sfreq).s(1) * 3 ' pure SHORT block
        cb_width = sfBandIndex(sfreq).s(1)
        cb_begin = 0

      Else
        next_cb_boundary = sfBandIndex(sfreq).l(1)  ' LONG blocks: 0,1,3
      End If
    ' apply formula per block type
      For sb = 0 To SBLIMIT - 1
```

```
For ss = 0 To SSLIMIT - 1
  If ((sb * 18) + ss = next_cb_boundary) Then ' Adjust critical band boundary
    If (gr_info.window_switching_flag And (gr_info.block_type = 2)) Then
      If (gr_info.mixed_block_flag) Then
        If (((sb * 18) + ss) = sfBandIndex(sfreq).l(8)) Then
          next_cb_boundary = sfBandIndex(sfreq).s(4) * 3
          cb = 3
        cb_width = sfBandIndex(sfreq).s(cb+1) -
          sfBandIndex((sfreq).s(cb))
        cb_begin = sfBandIndex(sfreq).s(cb) * 3
        End If
      ElseIf (((sb * 18) + ss) < sfBandIndex(sfreq).l(8)) Then
        next_cb_boundary = sfBandIndex(sfreq).l((++cb) + 1)
      Else
        next_cb_boundary = sfBandIndex(sfreq).s((++cb) + 1) * 3
        cb_width = sfBandIndex(sfreq).s(cb+1) -
        sfBandIndex(sfreq).s(cb)
        cb_begin = sfBandIndex(sfreq).s(cb) * 3
      End If
    Else
      next_cb_boundary = sfBandIndex(sfreq).s((++cb) + 1) * 3
      cb_width = sfBandIndex(sfreq).s(cb+1) -
      sfBandIndex(sfreq).s(cb)
```

```
            cb_begin = sfBandIndex(sfreq).s(cb) * 3
          End If
        Else ' long blocks
        next_cb_boundary = sfBandIndex(sfreq).l((++cb) + 1)
        End If
        ' Compute overall (global) scaling.
        xr(sb, ss) = Math.Pow(2.0, (0.25 * (gr_info.global_gain - 210.0)))
        ' Do long/short dependent scaling operations.
        If (gr_info.window_switching_flag And ( _
          ((gr_info.block_type = 2) And (gr_info.mixed_block_flag = 0)) Or _
          ((gr_info.block_type = 2) And gr_info.mixed_block_flag And (sb >= 2))))
Then
        xr(sb,ss) *= pow(2.0, 0.25 * -8.0 *  _
            gr_info.subblock_gain[(((sb*18)+ss) - cb_begin)/cb_width))
        xr(sb,ss) *= pow(2.0, 0.25 * -2.0 * (1.0+gr_info.scalefac_scale) _
            * (*scalefac)(ch).s((((sb*18)+ss) - cb_begin)/cb_width,cb))
          ' End If
        Else  ' LONG block types 0,1,3 & 1st 2 subbands of switched blocks *
        xr(sb,ss) *= pow(2.0, -0.5 * (1.0+gr_info.scalefac_scale) _
                * ((*scalefac)(ch).l(cb) _
                + gr_info.preflag * pretab(cb)))
        End If
        ' Scale quantized value.
```

```
            If (iis(sb, ss) < 0) Then
               sign = 1
            Else
               sign = 0
            End If
         xr(sb,ss) *= math.pow( convert.ToDouble( abs(iis[sb][ss])),
(convert.ToDouble(4.0/3.0)) )
         if (sign) xr(sb,ss) = -xr(sb,ss)
            End If
         Next
      Next
   End Sub
```

3.3.5.1 Header

The MDCT header based on the MP3 is 4 bytes long and within it is information about the layer type, the bitrate, the sampling frequency and its stereo mode as shown in Listing 3.7. There is also a 12-bit synchronized word used to find the start of a frame in a bitstream especially during playback, or can be used for broadcasting applications (Lagerström, 2001).

3.3.5.2 Side Information

Important information is located on the side information section. Information like the one necessary to decode the main data, information on Huffman table selection, the scale factors, and also the requantization parameters and windowing function selection

(Lagerström, 2001). Listing 3.8 shows the program code that is responsible for adding the extra information within the decoder. This side information section is 17 bytes long in single channel and 32 bytes in dual channel mode.

Listing 3.8: The Side information coding

```
Private Sub iteration_loop(ByVal mdct_freq_org(,,) As Double, ByRef side_info As side_info_t, ByVal enc(,,) As Integer, ByVal mean_bits As Integer, ByRef scalefactor As scalefac_t)

    Dim cod_info As gr_info
    Dim main_data_begin As Integer
    Dim scalefac_band_long = Peek(sfBandIndex(3).l(0))
    Dim max_bits As Integer
    Dim ch, gr, i As Integer
    Static firstcall As Integer = 1

    Dim xr(2, 2, samp_per_frame2) As Double
    'For i = 0 To 5
    'Dim l(23) As Integer
    'Dim s(14) As Integer
    '    sfBandIndex(i).l()
    '    sfBandIndex(i).s()
    'Next
    main_data_begin = Peek(side_info.main_data_begin)
```

```
        If (firstcall) Then
            main_data_begin = 0
            firstcall = 0
        End If
        scalefac_band_long = sfBandIndex(config.mpeg.samplerate_index +
(config.mpeg.type * 3)).l(0)
        For gr = 2 To 0 Step -1
            For ch = config.wave.channels To 0 Step -1
                For i = samp_per_frame2 To 0 Step -1
                    xr(gr, ch, i) = mdct_freq_org(gr, ch, i)
                Next i
            Next ch
        Next gr
        For gr = 2 To 0 Step -1
            For ch = config.wave.channels To 0 Step -1
                'cod_info = Peek(gr_info) And side_info.gr(gr).ch(ch)
                'calculation of number of available bit( per granule )
                max_bits = mean_bits / config.wave.channels
                '/* reset of iteration variables */
                'memset(scalefactor.l(gr, ch), 0, 22)
                'memset(scalefactor.s(gr, ch), 0, 14)
                For i = 4 To 0 Step -1
                    cod_info.slen(i) = 0
```

```
            Next i
            cod_info.part2_3_length = 0
            cod_info.big_values = 0
            cod_info.count1 = 0
            cod_info.scalefac_compress = 0
            cod_info.table_select(0) = 0
            cod_info.table_select(1) = 0
            cod_info.table_select(2) = 0
            cod_info.region0_count = 0
            cod_info.region1_count = 0
            cod_info.part2_length = 0
            cod_info.preflag = 0
            cod_info.scalefac_scale = 0
            cod_info.quantizerStepSize = 0.0
            cod_info.count1table_select = 0

            cod_info.quantizerStepSize = Convert.ToDouble(quantanf_init(xr(gr, ch,
samp_per_frame2)))
            cod_info.part2_3_length = outer_loop(xr, max_bits, enc, gr, ch, side_info)
            ResvAdjust(cod_info, mean_bits)
            cod_info.global_gain = cod_info.quantizerStepSize + 210
        Next ch
    Next gr
```

```
    ResvFrameEnd(side_info)
End Sub
```

3.3.5.3 Main Data

This section contains the Huffman coded frequency lines which is the "main data" and the coded scale factor values (Lagerström, 2001). The Huffman values are selected as per the criterion in Listing 3.6. The length depends on the bitrate and the length of the ancillary data.

Listing 3.9: Main data coding

```
Shared Function read_samples(ByVal sample_buffer As Short, ByVal frame_size As Integer) As Integer
    Dim samples_read As Integer
    Select Case (config.wave.type)
      Case Else
        MsgBox("[read_samples], wave filetype not supported")
        Case WAVE_RIFF_PCM
        samples_read = fstream(sample_buffer, sizeof(Of Short), frame_size, config.wave.file)
        ' we must swap if this is a big-endian machine
        If (config.byte_order = order_bigEndian) Then
          SwapBytesInWords(sample_buffer, samples_read)
        End If
        If (samples_read < frame_size And samples_read > 0) Then
```

```
            While (samples_read < frame_size)  sample_buffer(samples_read + 1) = 0
            End While
        End If
    End Select
    Return samples_read
End Function
```

The parameter responsible for indicating whether bits from previous frames are needed is the “main data begin”parameter in the side information. All the main data for one frame is stored in that and previous frames. The maximum size of the bit reservoir is 511 bytes (Lagerström, 2001).

3.3.5.4 Ancillary Data

This section is intended for user-defined data andtherefore is not specified further in the standard MP3 standard. It is not required in the process of compression of the audio data (Lagerström, 2001). This is a few unused bits that are used to fill up the frame up to the next byte boundary where the next frame starts.

CHAPTER FOUR

RESULTS AND DISCUSSION

4.1 The Interface

The different project modules that formed the various project development steps were all integrated in the Interface shown in Figure 4.1. These steps included capturing of raw uncompressed waveform samples, to MP3 encoding to the final MDCT Hybrid filter bank compression. Figure 4.1 shows a picture of the front panel of the MDCT encoder.

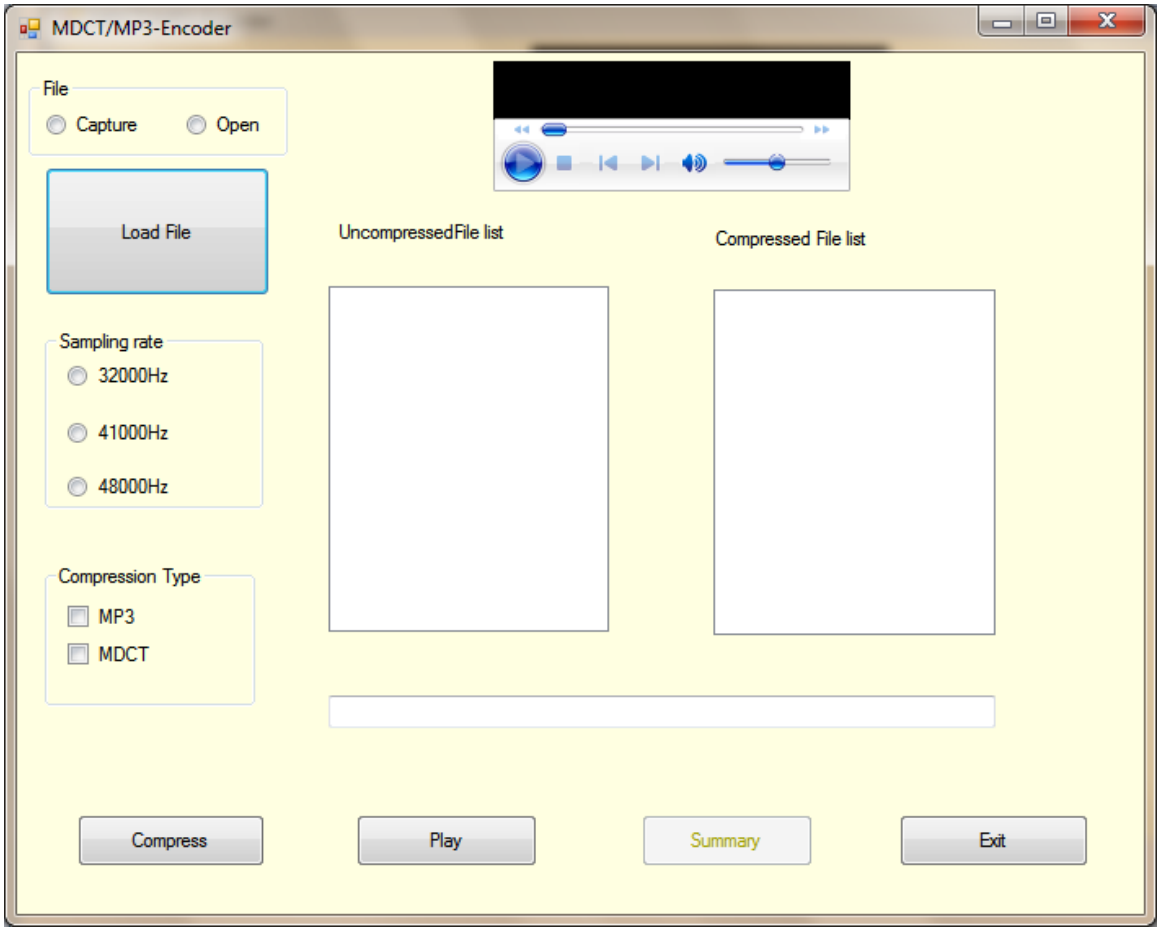

Figure 4.1: The MDCT project Front panel.

The encoder is divided into several sections described in the proceeding sections.

4.1.1 File Open/Capture

The file to be compressed is either some already recorded audio file in the hard disk by which case the "Open" Radio button at the top left corner of the interface is checked, or a file which is directly captured and sampled at a rate chosen from the three sampling rates 32000Hz, 41000Hz or 48000Hz as grouped together in figure 4.1. The Capture Radio button initiates the recording of a new audio from a sound recording device, which is then saved before compressing.

The window in Figure 4.2 is the recording interface. It contains buttons for Naming the raw file (File Name button). The "Device" button creates a list of the available recording devices. Once the recording device is selected by clicking the drop down button from the combobox below the date, then the file is given a name and stored at a chosen location accessed by clicking the "File Name" command. The recording begins when the "Start" command is clicked, and stopped with the "Stop" command. The application can be closed using the "Exit" button.

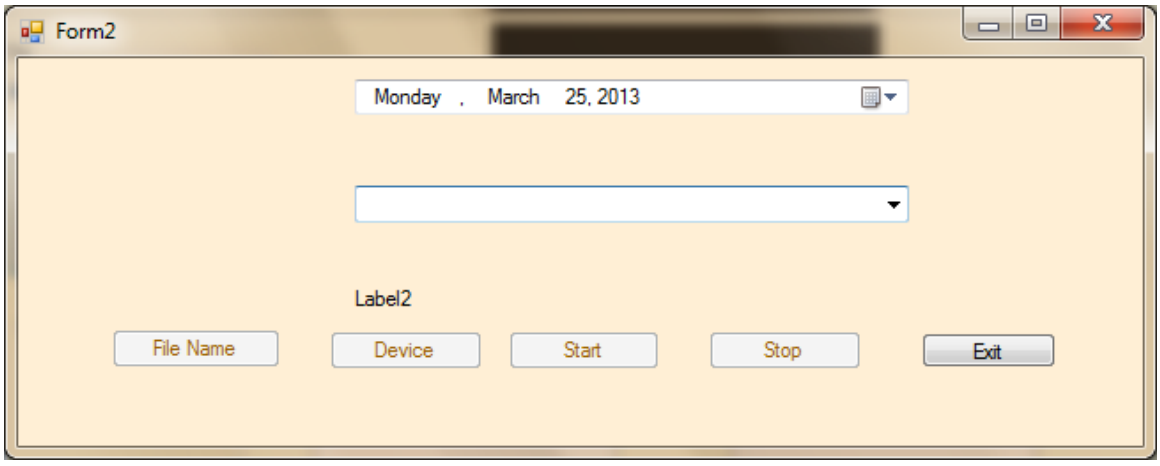

Figure 4.2: The recording interface

4.1.2 Sampling Rate

There are generally three sampling rates used in the MP3 encoding, which are 32,000Hz, 41,000Hz and 48,000Hz. The interface in figure 4.1 provides the option of choosing any of these rates by checking the respective Radio button.

4.1.3 Compression Type

Figure 4.1 provides options in form of checkboxes through which the audio file may be compressed by use of either the Standard MPEG or the MDCT laced MP3, or both. Checkboxes have been preferred over option buttons so that it makes it easy in the situation where the two compression methods need to be compared.

4.1.4 File Manipulation

The Command buttons at the bottom of the interface of figure 4.1 are for Compressing, playing, displaying file summary information and exiting the application respectively. The compression type has to be selected before the “Compression” button is clicked. The “Summary” button is activated as soon as the compression is complete. Clicking on it will display the summary of the number of bits that made up the original uncompressed audio file and the final bits after compression. These details are displayed as shown in figure 4.3. The WAV file, sampled at 41 KHZ is uncompressed while the MP3 and the MDCT files are compressed. As can be seen, the original file had 28,008,044 bits, which came down to 235,008 bits (91.63%) for the MDCT copression.

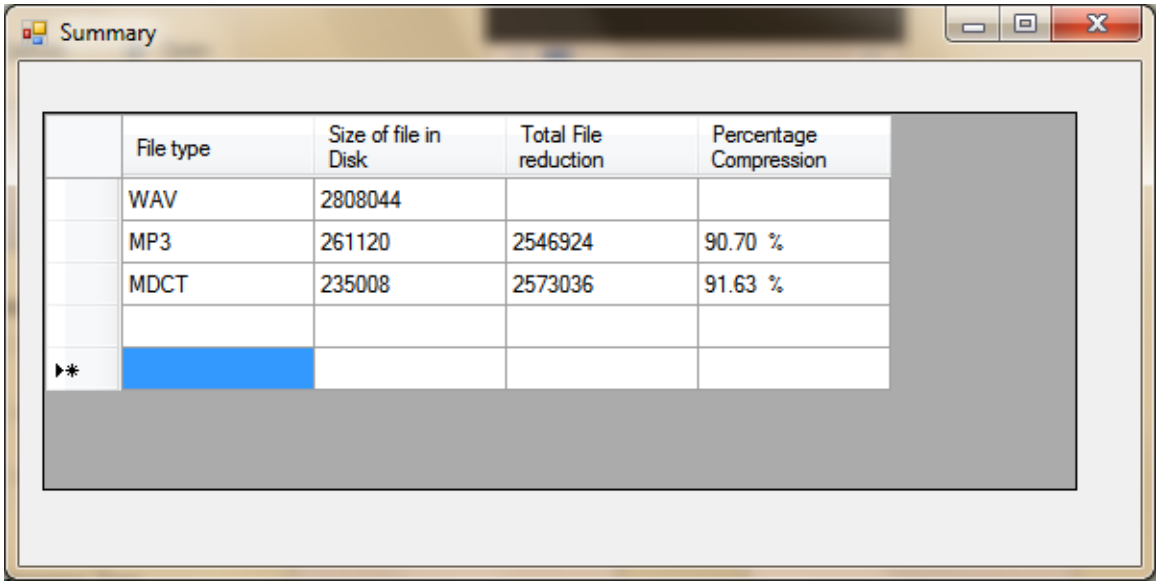

Figure 4.3: File compression summary window.

4.1.5 Miscellaneous

The Windows Media Player component at the top of figure 4.1 makes it easy to play, pause, rewind or stop an audio file while it is being played. The compressed files are displayed on the compressed file list and the uncompressed ones on the respective file list box.

4.2 Quantitative Analysis

Tables 4.1, 4.2 and 4.3 provide compression results for signals sampled at 32KHz, 41KHz and 48KHz respectively. The samples compressed were captured by Microphone at a bit rate of 128 kbps. A choice of any other sampling rate would affect the quality of the audio signal, but the general output would follow a similar pattern.

The uncompressed files in table 4.1 show file sizes that all end with 44 bits. These bits

represent the RIFF header that must proceed any data to be captured from input sources. Each signal therefore has an extra 44 bits that represent the header.

Table 4.1: Compression percentages for audio signals at a sampling rate of 32000Hz

AUDIO FILE	UNCOMPRESSED FILE SIZE ON DISK (BITS)	COMPRESSED FILE SIZE ON DISK (BITS)	COMPRESSION PERCENTAGE
AUDIO1_32.WAV	6560044	616464	90.60 %
AUDIO2_32.WAV	1376044	130464	90.52 %
AUDIO3_32.WAV	1472044	139104	90.55 %
AUDIO4_32.WAV	1616044	152928	90.54 %
AUDIO5_32.WAV	4160044	391392	90.59 %
AUDIO6_32.WAV	1568044	148176	90.55 %
AUDIO7_32.WAV	1616044	152928	90.54 %
AUDIO8_32.WAV	1456044	137808	90.54 %

Table 4.2: Compression percentages for audio signals at a sampling rate of 41000Hz

AUDIO FILE	UNCOMPRESSED FILE SIZE ON DISK (BITS)	COMPRESSED FILE SIZE ON DISK (BITS)	COMPRESSION PERCENTAGE
AUDIO1_41.WAV	1886044	185154	90.18 %
AUDIO2_41.WAV	1681044	165510	90.15 %
AUDIO3_41.WAV	1332544	131656	90.12 %
AUDIO4_41.WAV	2029544	199365	90.18 %

AUDIO5_41.WAV	1845044	181393	90.17 %
AUDIO6_41.WAV	2070544	203127	90.19 %
AUDIO7_41.WAV	1168544	115355	90.13 %
AUDIO8_41.WAV	2562544	251192	90.20 %
AUDIO9_41.WAV	1988544	195186	90.18 %
AUDIO10_41.WAV	2706044	265403	90.19 %

Table 4.3: Compression percentages for audio signals at a sampling rate of 48000Hz

AUDIO FILE	UNCOMPRESSED FILE SIZE ON DISK (BITS)	COMPRESSED FILE SIZE ON DISK (BITS)	COMPRESSION PERCENTAGE
AUDIO1_48.WAV	2544044	213120	91.62 %
AUDIO2_48.WAV	1704044	143232	91.59 %
AUDIO3_48.WAV	2880044	241152	91.63 %
AUDIO4_48.WAV	1824044	153216	91.63 %
AUDIO5_48.WAV	2304044	193152	91.62 %
AUDIO6_48.WAV	2472044	206976	91.63 %
AUDIO7_48.WAV	1968044	165120	91.61 %
AUDIO8_48.WAV	5976044	499200	91.65 %
AUDIO9_48.WAV	2256044	189312	91.61 %
AUDIO10_48.WAV	2232044	187008	91.62 %

From the three table results, it can be seen that there was just over 90% reduction in the compression when the MDCT filtering was applied. There 41 KHz audio compression in table 4.2 shows a more improved compression than both 32 KHz and 48 KHz. The reason for this is due to the fact that during the sampling process, there are rounding offs involved when dividing the samples of 41,000 into groups of 16 bits.

The other two sampling rates perfectly fit the 16 bit divisions. It is also observed that the higher the sampling rate the smaller the percentage improvement in the compression. This can be attributed to the fact that the more samples taken during the sampling process, the closer we are able to approximate our signals to the real analogue values.

The high perccentage compression is very significant in reducing huge files to small sizes that can easily be transmitted through digital media. Large uncompressed audio files of sizes amounting to Gigabytes now get sized down to tiny chunks of Kilobytes.

Table 4.4: Compression ratios for MP1, MP2 and MP3 compression techniques for audio signals at a sampling rate of 44100Hz (David and Price, 2012).

	MPEG LAYER1	MPEG LAYER2	MPEG LAYER3 (MP3)
Raw Data Rate (stereo at 44.1K samples per second	1.5 Mbps	1.5 Mbps	1.5 Mbps

Compressed Data Rate	384 kbps	192 kbps	128 kbps
Typical Compression	4:1	8:1	12:1

Table 4.4 shows the different types of audio compression used in MPEG systems and the relative amount of compression that they can provide (David and Price, 2012). This table uses a 2 channel stereo signal that is sampled at 44.1k samples per second, 16 bits per sample as a reference. The MPEG layer 1 coder can compress this signal to approximately 384 kbps (4:1 compression). The MPEG layer 2 coder can compress the signal to 192 kbps (8:1 compression). The MP3 coder can compress the signal to 128 kbps (12:1 compression). Comparred to the results obtained using the decoder developed in this project as shown in table 4.2; the compression ratios for MP3 sampled at 44.1 KHZ given as 12:1 from the cited material, closely match.

4.3 Qualitative Analysis

4.3.1 Subjective Testing

Audio playback test techniques that define the way listeners of audio output grade the quality of an audio signal in a rather structured manner, involve the assessing of the quality of audio signals according to some given grading scale that has been recommended by an official ITU standard ITU-R BS.1284 (Sporer, 1997). The document summarises the various ITU standards previously used by users to assess audio quality. A 5-point scale given in (Sporer, 1997) is shown in Table 4.5 with quality scores ranging from 1.0 to 5.0.

Table 4.5: Listening Tests Grading Scale based on ITU-R BS.1284 standard ranging from 1.0 to 5.0 (Sporer, 1997).

QUALITY		IMPAREMENT	
5.0	Excellent	5	Imperceptible
4.0	Good	4	Perceptible but not annoying
3.0	Fair	3	Slightly annoying
2.0	Poor	2	Annoying
1.0	Bad	1	Very annoying

One of the objectives of this project was to establish whether the MDCT encoding would produce a better compression without compromising on signal quality. This was achieved through the following method:

1. Generating an uncompressed wave audio file, called "*.wav". Where * is the name of the input file.
2. Compressing the input file into an MP3 using the MDCT hybrid filter.
3. Playing back the audio file using the Windows Media player component.

To gauge the quality of the compressed audio signal, the form shown in table 4.2 was designed and availed in soft copy to 20 volunteers. 15 students and 5 trainers all from Institute of advanced technology (IAT). Based on their listening abilities, they were required to choose whether the quality ranged from 96 % to 100%, 90% to 95%, 81% to 89 % or anything less than 80 %. Each volunteer was required to click on one of the four

options shown in table 4.6 after listening to both the raw uncompressed signal and the compressed MDCT signal.

Table 4.6: Listener satisfaction survey questionnaire.

Range in Percentage	Listener's Choice
96 – 100	20
91 – 95	None
86 – 90	None
80 – 85	None
Less than 80	None

All the volunteers clicked the 96% to 100% option, confirming that the quality of the compressed signal closely matched that of the original uncompressed one. In comparison, these results mapped to the ITU-R BS.1284 test grading system which would be viewed as table 4.7 below shows that the audio quality of the compressed audio can be concluded as excellent.

Table 4.7: Table showing comparrisons between the test results of the study, with the ITU-R BS.1284 grading system.

Percentage values	ITU-R BS.1284 Mapping
96 – 100	5.0
90 – 95	4.0
81 – 89	3.0
Less than 80	2.0
Less than 80	1.0

The compressed audio could not be differentiated from the original uncompressed file as attested by the number of listeners who volunteered to listen to the Playback.

Considering the Sampling rates available for audio sampling, there was a tradeoff between Audio Quality vs. Data Rate. The higher the bit rate, the higher the quality of the audio signal. However, Sampling rates do not affect file size as much as bit rates do. Bit rates of 128kbps were used in this project.

Furthermore, Audio1_32.wav sample audio file recorded as a raw uncompressed PCM was played back using the standard Media player packaged with Windows operating system, and the waveform in figure 4.4 was observed. It represents sound pressure observed against the time the music takes to play.

Figure 4.5 shows the waveform for the compressed MP3 file, Audio1_32.mp3 screen captured at the same time instance (after 20 seconds) as the previous sample.

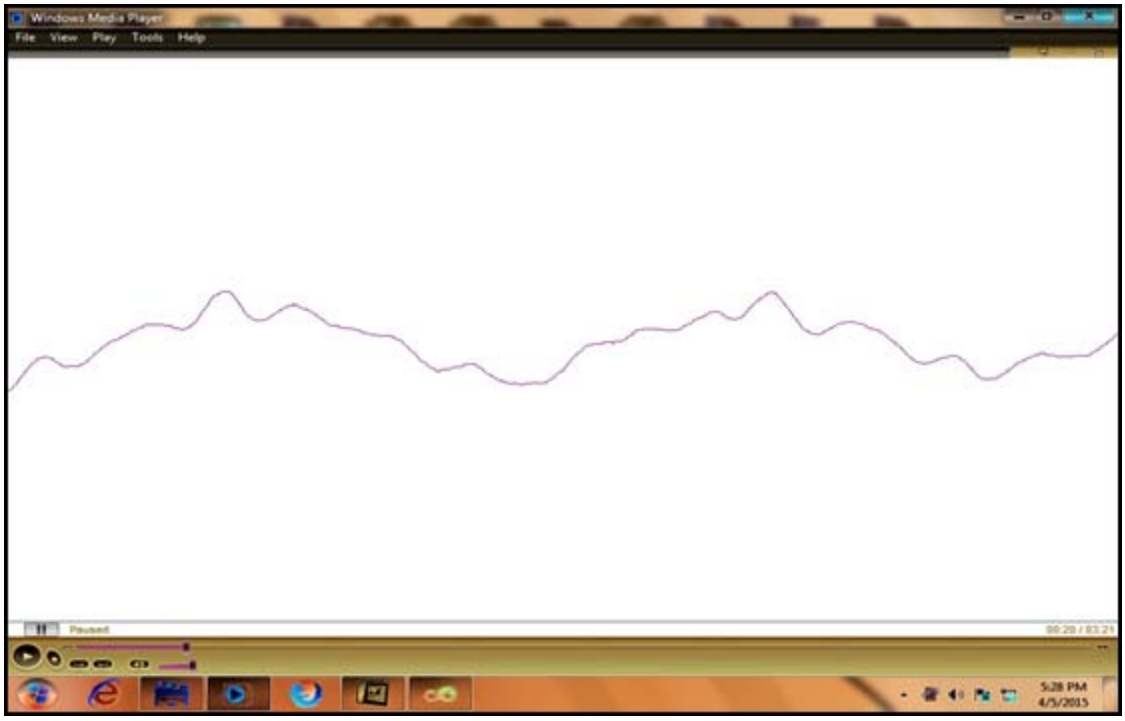

Figure 4.4: Audio1_32.wav waveform.

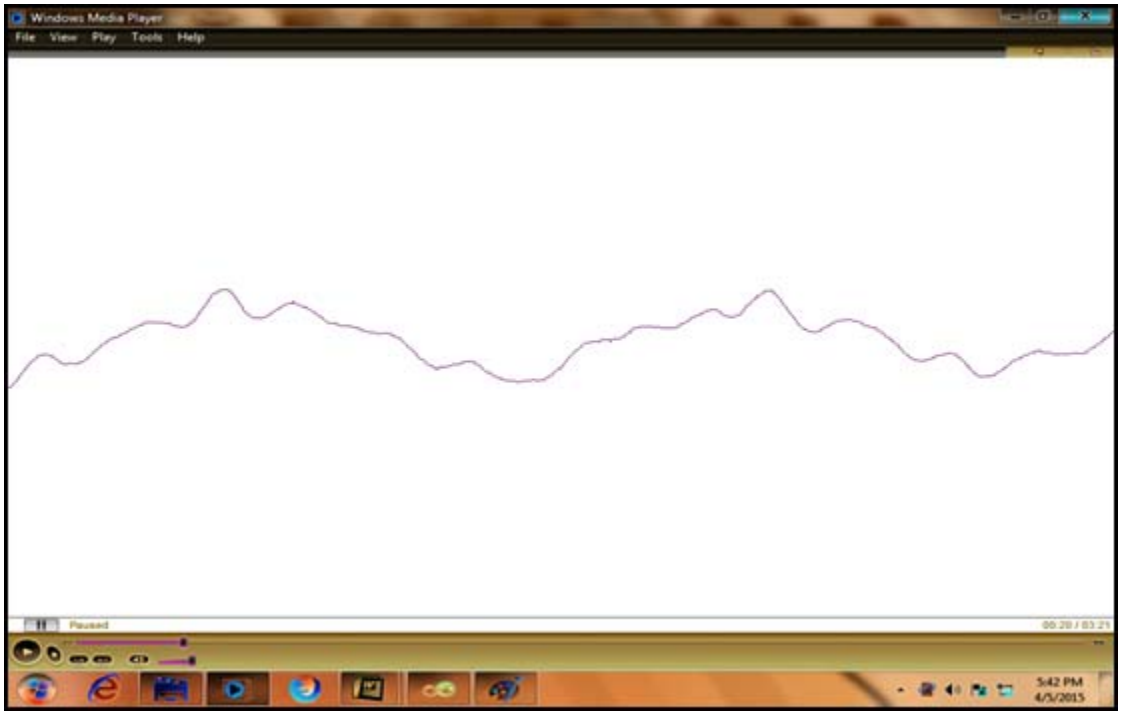

Figure 4.5: Audio1_32.mp3 waveform.

A detailed observation of the two waveforms in figure 4.4 and figure 4.5 are quite identical, which gives a strong indication that the two signals indeed sound the same

when listened to during playback.

Both figure 4.4 and figure 4.5 are screen captures captured after 20 seconds of playback time for both the uncompressed and the compressed audio file. The length of time it takes to playback both files is the same.

The similarities of the waveforms in figure 4.4 and figure 4.5 can better appreciated through the combination area graph in figure 4.6. The section Labeled A represents the waveform of figure 4.4, while the area labeled B represents the waveform in figure 4.5.

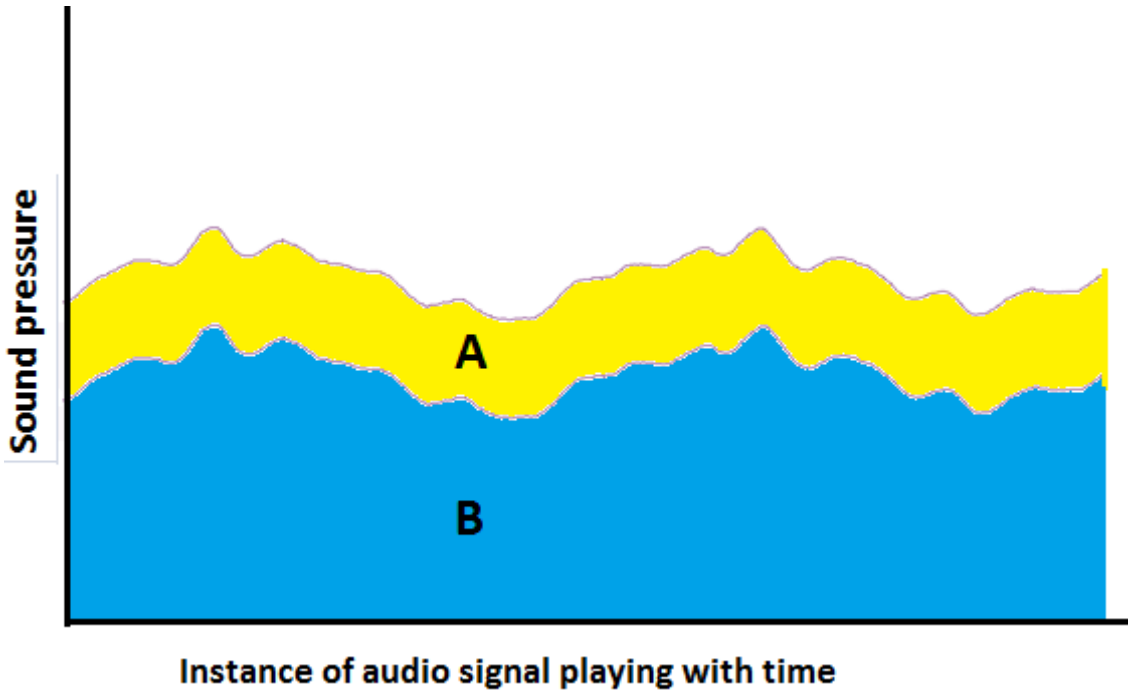

Figure 4.6: Area graph showing combinations of the waveforms in Figure 4.4 and Figure 4.5.

It should be noted that the similarities observed is a confirmation that despite the loss of uneccesary samples in the compressed audio signal, the fidelity of the final file is quite close to the CD quality PCM audio file.

4.2.2 Objective Testing

The subjective testing technique is generally time consuming and expensive. The objective testing technique uses a cognitive model in PEAQ which imitates the cognitive processing system of the human brain. The human brain cognitive process is used to compute a quality score for audio signals. The cognitive model processes in PEAQ use the parameters that are produced by the psychoacoustic human ear models to generate output values known as Model Output Variables (MOVs). Following the production of the MOVs, the values are mapped to a single Objective Difference Grade (ODG) score. There are 11 MOVs associated with the Basic Version and 5 MOVs associated wih the Advanced Version. These MOV values eventually enter a Multi-Layer Perceptron Neural Network (MLPNN). The training of the neural network software produces the ODG scores. The training itself requires that a large collection of human subjective data values obtained from listening tests (Campbell, 2009).

The PEAQ audio assesment model produces MOVs which are as result of a comparison between a reference signal (PCM for our case) and the same signal processed by particular codec like compressing it using MDCT. Lastly the PEAQ model brings together the MOVs to produce one value as output that corresponds to an expected output result from a subjective quality assessment ODG score (Thiede, 2000). The ODG is the

output variable obtained from the objective measurement method that use used in PEAQ and it corresponds to a Subjective Difference Grade (SDG) in the subjective testing method.

The SDG value obtained from the analysis of the results from the subjective listening test is defined as:

SDG = Grade Signal under test – Grade Reference signal (Salovarda, 2004).

Table 4.8 gives the relationship between the subjective gradings of Table 4.7 with the SDGs.

Table 4.8: SDG description

Impairment	**Grade**	**SDG**
Imperceptible	5.00	0.00
Perceptible but not annoying	4.00	-1.00
Slightly annoying	3.00	-2.00
Annoying	2.00	-3.00
Very annoying	1.00	-4.00

In this study, ten audio Reference sample files, named AUDIO1_48.WAV, AUDIO2_48.WAV all the way to AUDIO10_48.WAV, were used. They are all WAV-files sampled at 48 kHz, 16-bit PCM. Each has a length of approximately 10 seconds. The corresponding Degraded outputs AUDIO1_48.MP3, AUDIO2_48.MP3 all the way to AUDIO10_48.MP3 respectively, are the compressed MP3 audio files whose quality

was evaluated using PEAQ algorithm. Table 4.9 Shows the OGV values obtained from the evaluation.

Table 4.9: Objective Difference Grade (ODG) scores for ten audio samples sampled at 48 kHz, 16-bit PCM

Reference WAV sample	Degraded MP3 sample	ODG values
AUDIO1_48.WAV	AUDIO1_48.MP3	-0.616
AUDIO2_48.WAV	AUDIO2_48.MP3	-0.540
AUDIO3_48.WAV	AUDIO3_48.MP3	-1.792
AUDIO4_48.WAV	AUDIO4_48.MP3	-0.407
AUDIO5_48.WAV	AUDIO5_48.MP3	-1.102
AUDIO6_48.WAV	AUDIO6_48.MP3	-0.571
AUDIO7_48.WAV	AUDIO7_48.MP3	-1.871
AUDIO8_48.WAV	AUDIO8_48.MP3	-0.886
AUDIO9_48.WAV	AUDIO9_48.MP3	-0.844
AUDIO10_48.WAV	AUDIO10_48.MP3	-1.771

It is visible from table 4.5 that all the ODG values for all the ten samples are within the "slightly annoying" to the "imperceptible" ranges as compared to the SDGs of table 4.8. From these results we can conclude that the quality of the compressed audio file is fairly close to that of the uncompressed WAV file.

4.2.3 Objective Tests results from other works

Table 4.91 shows the results of tests previously taken from various codecs by the use of the PEAQ measurement software as recommended by the ITU-R BS.1387. These codecs are the standard MP2 and MP3 Lame (also named as MPEG 1 layer 2 and layer 3 respectively, based on ISO/IEC 11172/3, 1992. ISO/IEC stands for International Standards Organization/International Electrotechnical Commission), Advanced Audio Coding (AAC), according to ISO/IEC 13818/3, 1994.) And OGG Vorbis from Xiph.org, and is different from MPEG 1 and 2 standards (Salovarda *et al*, 2004).

All the codecs aforementioned use perceptual coding or psychoacoustic methods to perform lossy compression. The differences among them is therefore in their algorithms and and on how complex they are. They do not behave in a similar manner as can be observed from the results of the measurements (Salovarda *et al*, 2004).

Table 4.91: ODG values and file size for four codecs on most common bit rates (Salovarda et al, 2004).

BIT RATES	**CODEC**	**ODG**	**FILE SIZE**
32 kbps	MP2	-3.85	56 KB
	MP3	-3.67	56 KB
	AAC		
	OGG		
64 kbps	MP2	-3.28	111 KB

	MP3	-3.46	112 KB
	AAC	-3.36	111 KB
	OGG	-3.24	113 KB
128 kbps	MP2	-2.36	222 KB
	MP3	-1.08	223 KB
	AAC	-1.09	222 KB
	OGG	-0.34	221 KB
160 kbps	MP2		
	MP3	-0.47	276 KB
	AAC	-0.41	278 KB
	OGG	-0.20	271 KB
192 kbps	MP2	-0.59	332 KB
	MP3	-0.16	334 KB
	AAC	-0.20	333 KB
	OGG	-0.09	326 KB
256 kbps	MP2	-0.19	443 KB

	MP3	-0.01	445 KB
	AAC	-0.12	429 KB
	OGG	0.02	460 KB
320 kbps/ 350kbps -ogg	MP2		
	MP3	0.04	556 KB
	AAC	-0.02	535 KB
	OGG	0.07	632 KB

From the results shown in table 4.91, all the codecs have a similar behavior at bit rates greater than 160 kbps. The differences are very small except for the MP2. When the bit rates are less than 160 kbps on the alternate, we then observe different behavior of all the four given codecs, especially in the most interesting case of 128 kbps. The best at this rate is OGG Vorbis, very similar qualities are observed for AAC and MP3, MP2 has the lowest quality at this bit rate (Salovarda *et al*, 2004).

CHAPTER FIVE

CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

In this study, it has been demonstrated that MDCT encoding process provides a further reduction in the number of bits that can be used to represent an audio signal without affecting the listening experience by listeners. The PEAQ encoding technique employs the technique by observing and analyzing the human ear and the auditory system, all the while trying to refine the psychoacoustic model, and at the same time allowing the model to mathematically determine what information can be discarded from the data. The final result is an encoding technique that allows for very high compression ratios compared to its lossless counterparts.Both subjective and objective evaluations of the compressed audio files in this study were performed using Listening tests and using Perceptual Evaluation of Audio Quality (PEAQ) algorithm respectively.

The advantage to this is that the increased demand for multimedia, cinema, and home theater applications can take advantage of reduced disk space and increased quality testing and transmission speeds as a result. Results of PEAQ software for objective audio quality measures for the prediction of the perceived quality of compressed files and the Quality measures for the subjective tests were compared, and they showed high correlations. PEAQ results obtained by the measurement of ODGs indicated that these objective quality measures are reliable and can be used as tools for measures of audio quality evaluation of commercial codecs and music productions.

5.2 Recommendation for future work

Future work may include looking at how to improve on the implementation of the Digital signal Processor, and the bus speeds and in general, the hardware implementations of the decoder. Research is being carried out to find out the potential improvements that can be made to the various perceptual coders and the application of psychoacoustics processes and building blocks, such as novel filter banks for low-delay coding and reduced pre-echo (Schuller, 1998), and thus new psychoacoustic signal analysis methods and techniques are still in the process of being investigated. An exploration into the application of the Modified discrete Sine transform (MDST) algorithm is recommended so that the complex elements of audio signals are also considered in audio compression techniques.

REFERENCES

Arjona, M. Ramírez, M. (2008). Technology and standards for low-bit-rate coding methods. *In The Handbook of Computer Networks*, **2:** 447–467.

Brandenburg, K. (1987). Evaluation of quality for audio encoding at low bit rates. *Contribution to the 82nd AES Convention, preprint 2433. London, United Kingdom. 100 Rec. ITU-R BS.1387-1*.

Brandenburg, K. and Stoll G. (2009). ISO-MPEG-1 Audio: A Genereic Standard for Coding of High Quality Digital Audio. *Journal of the Audio Engineering Society*, **10**: 780-792.

Bosi, M. (1997). ISO/IEC MPEG-2 Advanved Audio Coding, *Journal of the Audio Engineering Society*, **10**: 789-813.

Campbell,D, Edward, J. and Martin, G. (2009). National University of Ireland, Galway. *Audio Quality Assessment Techniques – A Review, and Recent Developments*

Chiariglione, L. (2009). Riding the Media Bits. MPEG's first Steps.

Davis, P. (2005). A Tutorial on MPEG/Audio Compression. *IEEE Multimedia systems and applications*, **2**(2): 60-74.

David, L. and Price, H. (2012). MPEG codec comparrison. In *Introduction to MPEG* (p. 19). Althos.

Rao, K. and Yip, P. (1990), Discrete Cosine Transform. In *Wikipedia*. Retrieved May 15, 2012 from *http://en.wikipedia.org/wiki/Discrete_cosine_transform*

Dobson, K., Yang, J., Smart, K., and Guo, K. (1997). High Quality Low Complexity Scalable Wavelet Audio Coding, *in Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, **97**: 327-330

Jacaba J. (2001). Department of Mathematics College of Science, The University of the Philippines Diliman, Quezon City. *Audio compression using modified discrete cosine transform*.

Junyong, Y., Ulrich, R., Miska, M., Hannuksela, B., Moncef, G. and Andrew, P. (2010). Perceptual-based quality assessment for audio–visual services: *A survey*

Lagerstrom, K. (2001). Chalmers University of Technology, Department of Computer Engineering Gothenburg, Sweden. *Design and Implementation of an MPEG-1 Layer III Audio Decoder*.

Martin, E.S.G., Carotti, J.C.D. and A.R.Meo. (1992). Backward-adaptive loss-less compression of video sequences, *Proceedings of the IEEE International Conference on Audio, Speech and Signal Processing*, **4**: 3417.

Narasimha, M. and Peterson, A. (1978). On the Computation of the Discrete Cosine Transform. *IEEE Transactions on Communications*, **6**: 934–936.

Painter, T. and Spanias, A. (2000). Perceptual Coding of Digital Audio. *Proceedings of the IEEE*, **88(**4): 451-513.

Painter, T. and Spanias, A. (2000). A Review of Algorithms for Perceptual Coding of Digital Audio Signal. Retrieved on March 11, 2012 from http://www.eas.asu.edu/~speech/ndtc/dsp97.ps

Pan, D. (1995). A Tutorial on MPEG/Audio Compression. *IEEE Multimedia systems and applications*, **2**(2) 60-74.

Paul S. (2000). Perceptual Coding: How Mp3 Compression Works. Retrieved on June 20th 2012 from http://www.soundonsound.com/sos/may00/articles/mp3.htm.

Princen, J. and Bradley, A. (1986). Analysis/Synthesis Filter Bank Design Based on Time Domain Aliasing Cancellation. *IEEE Transactions, ASSP*, **34**(5): 1153-1161.

Princen, J., Johnson, A. and Bradley, A. (1987). Subband/Transform Coding Using Filter Bank Designs Based on Time Domain Aliasing Cancellation. *Proceedings of the International Conference on Audio, Speech and Signal Processing*, **34**(5): 2161-2164.

Šalovarda, M., Ivan Bolkovac, I. and Domitrovic, H. (2004). Faculty of Electrical Engineering and Computing, Unska 3, HR – Zagreb, Croatia, *Estimating Perceptual Audio System Quality Using PEAQ Algorithm.*

Salovarda,M., Bolkovac, I. and Domitrovic, H. (2004). Comparison of audio codecs using PEAQ algorithm, CUC04, Zagreb.

Schuller, G. (1998). Time-Varying Filter Banks with Low Delay for Audio Coding. *In Proceedings 105th Convention on Audio Engineers Society*, preprint No. 4809.

Sporer, T. (1997). Objective audio signal evaluation ñ applied psychoacoustics for modeling the perceived quality of digital audio. *In 103rd AES-Convention,* preprint 4512. New York, United States of America.

Sporer, T., Brandenburg, K. and Edler, B. (1997). The Use of Multirate Filter Banks for Coding of High Quality Digital Audio. *6th European Signal Processing Conference (EUSIPCO), Amsterdam*, **1:** 211-214.

Terhardt, E. (1979). Calculating Virtual Pitch, Hearing. *Research*. **1:** 155-182.

Thiede, T. (2000). PEAQ The ITU Standard for Objective Measurement of Perceived Audio Quality, *Journal of Audio Engineering Society,* **48**(1,2): 3-29.

Vanam R. and Creusere, C.(2005). Evaluating Low Bitrate Scalable Audio Quality Using Advanced Version of PEAQ and Energy Equalization Approach, *Proceedings of IEEE ICASSP*, **3:** 189-192.

Wen-Hsiung C., Smith, C., Fralick, S. (1977). A Fast Computational Algorithm for the Discrete Cosine Transform. *IEEE Transactions on Communications,* **9:** 1004–1009

William, D. and Madisetti, V. (1998). The Digital Signal Processing Handbook. *CRC Press*. **3**(2): 38.1 – 44.8.

APPENDICES

The Code

```
'*************************************************************************
Portions of the code used to build the MDCT coder in this study and its components
were adopted from the following sources:
    1. www.mpeg1.de
    2. Forum.gpwiki.org
    3. Read.pudn.com
    4. Lists.cs.uiuc.edu
    5. Luanvan.co
    6. Osm.ee.qub.ac.uk
    7. www.net118.com
    8. 128.59.11.212
    9. www1.cs.columbia.edu
    10. www.cnblogs.com
    11. www.elec.ryukoku.ac.jp
    12. Mediaplayer.limewire.org
    13. Forum.dbpoweramp.com
    14. www.mpeg2.de
    15. Scholar.havard.edu
    16. Arch.usc.edu
    17. www.hivmr.com
    18. Web.nuu.edu.tw
    19. Unix01.sac.edu
    20. www.vbfrance.com
    21. Lesbos.esat.kuleuven.ac.be
    22. Writing Add-ins for visual basics (publication)
    23. www.dotnetspark.com
    24. www.morikita.co.jp
    25. Moving to VB NET strategies and concepts (publication)
    26. Thesis.binus.ac.id
    27. Mro.massey.ac.nz
    28. www.aspnet-answers.com
    29. Data entry and validation with C# and VB.NET (publication)
    30. Pastebin.com
    31. Fr.slideshare.net
    32. Xn-im-5y2d.cn
    33. Vbex.ru
    34. Telin.rug.ac.be
    35. Internationalization and Localization Using Microsoft .NET (publication)
    36. www.techsupportforum.com
    37. smdn.jp
    38. www.580d0c.com
    39. Scripter.co.kr
    40. Peer-to-peer with VB NET, 2003 (publication)
    41. www.codeproject.com
    42. www.vnote.kr
    43. www.slideshare.net
    44. Sixeightone.eu
    45. www.notsayfam.com
    46. www.lob.cn
    47. Weblogs.asp.net
    48. www.visualstudiodev.com
    49. Instrunet.tistory.com
    50. www.ehow.com
```

```
*************************************************************************

Imports System.IO
Public Class MDCT
    Dim ThePath As String

    Private Sub CmdLoadUncompressed_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles CmdLoadUncompressed.Click
        Dim Extension As String
        If RadioOpen.Checked Then
            'CmdLoadUncompressed.Text = "Open Existing File"
            OpenFileDialog1.ShowDialog()
            Extension =
Microsoft.VisualBasic.Right(OpenFileDialog1.SafeFileName(), 4)
            If Extension = ".mp3" Then
                LstCompressed.Items.Add(OpenFileDialog1.SafeFileName())
            Else
                'CmdLoadUncompressed.Text = "Capture File"
                LstUncompressed.Items.Add(OpenFileDialog1.SafeFileName())
            End If
            TextBox1.Text = OpenFileDialog1.FileName()

        ElseIf RadioCapture.Checked Then
            If RadioButton1.Checked = False And RadioButton2.Checked = False And
RadioButton3.Checked = False Then
                MsgBox("Please select a sampling rate") : Exit Sub
            End If
            Form2.Show()
        Else : MsgBox("Please Choose File Capture or File open button.")
        End If
    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
        Dim ResourceFilePath As String
        If System.Diagnostics.Debugger.IsAttached() Then

            ResourceFilePath =
System.IO.Path.GetFullPath(OpenFileDialog1.FileName())

        Else

            ResourceFilePath = Application.StartupPath & "\resources\"

        End If


        MediaPlayer1.URL = TextBox1.Text 'ResourceFilePath & "\Today.mp3"

    End Sub


    Private Sub Button4_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)
        My.Computer.Audio.Stop()
    End Sub
```

```
    Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button3.Click
        End
    End Sub


    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

        Dim file = TextBox1.Text
        Dim newfile As String


        If CheckBox1.Checked And CheckBox2.Checked = False Then
            If Microsoft.VisualBasic.Right(TextBox1.Text, 4) = ".mp3" Then
                MsgBox("The file you are trying to compress is already in MP3
format")
                Exit Sub

            Else

                MsgBox("Ready to perform MP3 compression.")
            End If
            Try
                Shell(ThePath & file & " " & Microsoft.VisualBasic.Left(file,
Len(file) - 4) & ".mp3")
                newfile = Microsoft.VisualBasic.Left(file, Len(file) - 4) & ".mp3"
                LstCompressed.Items.Add(newfile)
                MsgBox("MP3 Compress complete")
            Catch ex As Exception
                MsgBox("Could not save input file!", vbExclamation + vbOKOnly,
My.Application.Info.Title)
                Exit Sub
                'Finally
                'MsgBox("MP3 Compress complete")
            End Try
        ElseIf CheckBox2.Checked And CheckBox1.Checked = False Then
            If Microsoft.VisualBasic.Right(TextBox1.Text, 4) = ".mp3" Then
                MsgBox("The file you are trying to compress is already in MP3
format")
                Exit Sub

            Else

                MsgBox("Ready to perform MDCT compression.")
            End If
            Try
                Shell(ThePath & file & " " & Microsoft.VisualBasic.Left(file,
Len(file) - 4) & ".mp3")
                newfile = Microsoft.VisualBasic.Left(file, Len(file) - 4) & ".mp3"
                LstCompressed.Items.Add(newfile)
                MsgBox("MDCT Compress complete")

            Catch ex As Exception
                MsgBox("Could not save input file!", vbExclamation + vbOKOnly,
My.Application.Info.Title)
                Exit Sub
                'Finally
```

```
            End Try
        ElseIf CheckBox1.Checked And CheckBox2.Checked Then
            If Microsoft.VisualBasic.Right(TextBox1.Text, 4) = ".mp3" Then
                MsgBox("The file you are trying to compress is already in MP3
format")
                Exit Sub

            Else

                MsgBox("Ready to perform both MP3 and MDCT compression.")
            End If
            Try
                Shell(ThePath & file & " " & Microsoft.VisualBasic.Left(file,
Len(file) - 4) & ".mp3")
                newfile = Microsoft.VisualBasic.Left(file, Len(file) - 4) & ".mp3"
                LstCompressed.Items.Add(newfile)
                MsgBox("Compression complete")

            Catch ex As Exception
                MsgBox("Could not save input file!", vbExclamation + vbOKOnly,
My.Application.Info.Title)
                Exit Sub
                'Finally
            End Try
        Else
            MsgBox("Yo haven't chosen a compression Type.") : Exit Sub
        End If
        Me.Button4.Enabled = True
    End Sub

    Private Sub Button4_Click_1(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button4.Click
        Dim file = TextBox1.Text
        LstCompressed.SelectedIndex = 0
        Dim fileMp3 = LstCompressed.Text
        Dim Mp3bytes As Integer
        Dim wavefile As FileStream
        Dim Mp3file As FileStream
        Dim MDCTfile As FileStream
        Dim value As Integer = 4 'Number of rows in the summary table
        Try
            wavefile = IO.File.Open(file, IO.FileMode.Open,
IO.FileAccess.ReadWrite)
            Summary.Show()
            Summary.DataGridView1.Rows.Add(value)
            Summary.DataGridView1.Rows(0).Cells(0).Value = "WAV"
            Summary.DataGridView1.Rows(1).Cells(0).Value = "NON-MDCT"
            Summary.DataGridView1.Rows(2).Cells(0).Value = "MDCT"
            Summary.DataGridView1.Rows(0).Cells(1).Value = wavefile.Length
            If CheckBox1.Checked And CheckBox2.Checked = False Then
                Mp3file = IO.File.Open(fileMp3, IO.FileMode.Open,
IO.FileAccess.ReadWrite)
                Mp3bytes = CInt(1 / 9 * Mp3file.Length)
                Summary.DataGridView1.Rows(1).Cells(1).Value = Mp3file.Length +
Mp3bytes
                Summary.DataGridView1.Rows(1).Cells(2).Value = wavefile.Length -
(Mp3file.Length + Mp3bytes)
```

```
                Summary.DataGridView1.Rows(1).Cells(3).Value =
FormatNumber(((wavefile.Length - (Mp3file.Length + Mp3bytes)) / wavefile.Length) *
100, 2) & "  %"

                Mp3file.Close()
            ElseIf CheckBox2.Checked And CheckBox1.Checked = False Then
                MDCTfile = IO.File.Open(fileMp3, IO.FileMode.Open,
IO.FileAccess.ReadWrite)
                Summary.DataGridView1.Rows(2).Cells(1).Value = MDCTfile.Length
                Summary.DataGridView1.Rows(2).Cells(2).Value = wavefile.Length -
MDCTfile.Length
                Summary.DataGridView1.Rows(2).Cells(3).Value =
FormatNumber(((wavefile.Length - MDCTfile.Length) / wavefile.Length) * 100, 2) & "
%"

                MDCTfile.Close()
            ElseIf CheckBox1.Checked And CheckBox2.Checked Then
                Mp3file = IO.File.Open(fileMp3, IO.FileMode.Open,
IO.FileAccess.ReadWrite)
                Mp3bytes = CInt(1 / 9 * Mp3file.Length)
                Summary.DataGridView1.Rows(1).Cells(1).Value = Mp3file.Length +
Mp3bytes
                Summary.DataGridView1.Rows(1).Cells(2).Value = wavefile.Length -
(Mp3file.Length + Mp3bytes)
                Summary.DataGridView1.Rows(1).Cells(3).Value =
FormatNumber(((wavefile.Length - (Mp3file.Length + Mp3bytes)) / wavefile.Length) *
100, 2) & "  %"

                Mp3file.Close()
                MDCTfile = IO.File.Open(fileMp3, IO.FileMode.Open,
IO.FileAccess.ReadWrite)
                Summary.DataGridView1.Rows(2).Cells(1).Value = MDCTfile.Length
                Summary.DataGridView1.Rows(2).Cells(2).Value = wavefile.Length -
MDCTfile.Length
                Summary.DataGridView1.Rows(2).Cells(3).Value =
FormatNumber(((wavefile.Length - MDCTfile.Length) / wavefile.Length) * 100, 2) & "
%"

                MDCTfile.Close()
            End If
            wavefile.Close()



        Catch ex As Exception
            MsgBox(ex.ToString)
        End Try

    End Sub

    Private Sub MDCT_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        Me.Button4.Enabled = False
    End Sub


    Private Sub RadioOpen_CheckedChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles RadioOpen.CheckedChanged
```

```
        ' If radio Then
        CmdLoadUncompressed.Text = "Open Existing File"
        'End If
    End Sub

    Private Sub RadioCapture_CheckedChanged(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles RadioCapture.CheckedChanged
        CmdLoadUncompressed.Text = "Capture File"
    End Sub
End Class
```

```
Imports System.IO
Imports System.Threading
Imports Microsoft.DirectX
Imports Microsoft.DirectX.DirectSound
Public Class Form2
    Inherits System.Windows.Forms.Form
    Private Devices As CaptureDevicesCollection
    Private device As Device
    Private info As DeviceInformation
    Public CaptureDeviceGuid As Guid = Guid.Empty
    Public ApplicationDevice As Capture = Nothing
    Public CaptureBufferSize As Integer = 0
    Public InputFormat As WaveFormat
    Private applicationBuffer As CaptureBuffer = Nothing
    Public Const NumberRecordNotifications As Integer = 16
    Public PositionNotify As BufferPositionNotify() = New
Microsoft.DirectX.DirectSound.BufferPositionNotify(NumberRecordNotifications + 1)
{}
    Public NextCaptureOffset As Integer = 0
    Private NotifyThread As Thread = Nothing
    Private Capturing As Boolean = False
    Private Recording As Boolean = False
    Public NotificationEvent As AutoResetEvent = Nothing
    Public NotifySize As Integer = 0
    Public applicationNotify As Notify = Nothing
    Private FileName As String = String.Empty
    Private datasum As Double = 0
    Private datasum2 As Double = 0
    Private WaveFile As FileStream = Nothing
    Private datafile As FileStream = Nothing
    Private Writer As BinaryWriter = Nothing
    Private Writer2 As StreamWriter = Nothing
    Private Reader As BinaryReader = Nothing
    Public SampleCount As Integer = 0
    Public timelength() As Double = New Double(100000) {}
    Public startTime As New DateTime
    Public EndTime As New DateTime
    Private Path As String = String.Empty
    Private n As Long = 0


    Private Sub Form2_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
```

```
        Devices = New CaptureDevicesCollection
        For Each info As DeviceInformation In Devices
            ComboBoxDevice.Items.Add(info.Description)
        Next info
        ButtonFileName.Enabled = False
        ButtonStart.Enabled = False
        ButtonStop.Enabled = False
        ButtonDevice.Enabled = False
        MsgBox("Select recording device from the combobox")

    End Sub

    Private Sub CreateCaptureBuffer()
        Dim CBDescription As CaptureBufferDescription
        CBDescription = New CaptureBufferDescription
        If applicationNotify IsNot Nothing Then
            applicationNotify.Dispose()
            applicationNotify = Nothing
        End If
        If MDCT.RadioButton1.Checked Then
            InputFormat.SamplesPerSecond = Val(MDCT.RadioButton1.Text)
        ElseIf MDCT.RadioButton2.Checked Then
            InputFormat.SamplesPerSecond = Val(MDCT.RadioButton2.Text)
        ElseIf MDCT.RadioButton3.Checked Then
            InputFormat.SamplesPerSecond = Val(MDCT.RadioButton3.Text)
        End If
        InputFormat.FormatTag = WaveFormatTag.Pcm
        'InputFormat.SamplesPerSecond ='48000
        InputFormat.BitsPerSample = 16
        InputFormat.Channels = 2
        InputFormat.BlockAlign = Int((InputFormat.Channels *
(InputFormat.BitsPerSample / 8)))
        InputFormat.AverageBytesPerSecond = InputFormat.BlockAlign *
InputFormat.SamplesPerSecond

        If InputFormat.AverageBytesPerSecond / 8 < 1024 Then
            NotifySize = 1024
        Else
            NotifySize = InputFormat.AverageBytesPerSecond / 8
        End If
        NotifySize -= NotifySize Mod InputFormat.BlockAlign 'make sure the buffer
is aligned
        CaptureBufferSize = NotifySize * NumberRecordNotifications
        CBDescription.BufferBytes = CaptureBufferSize
        InputFormat.FormatTag = WaveFormatTag.Pcm
        CBDescription.Format = InputFormat

        Try
            'MsgBox("Capture buffer succsessfully created!")
        Catch ex As Exception
            MsgBox(ex.ToString)
            'Me.Close()
        End Try
        applicationBuffer = New CaptureBuffer(CBDescription, ApplicationDevice)
        NextCaptureOffset = 0
        InitNotifications()
    End Sub
    Private Sub InitNotifications()
```

```
        '-------------------------------------------------------------------------
----
        ' Name: InitNotifications()
        ' Desc: Inits the notifications on the capture buffer which are handled
        ' in the notify thread.
        '-------------------------------------------------------------------------
----

        If applicationBuffer Is Nothing Then
            Throw New NullReferenceException()
        End If

        ' Create a thread to monitor the notify events
        If NotifyThread Is Nothing Then
            NotificationEvent = New AutoResetEvent(False)
            NotifyThread = New Thread(New ThreadStart(AddressOf WaitThread))

            ' Create a notification event, for when the sound stops playing

            Capturing = True
            NotifyThread.Start()
        End If

        ' Setup the notification positions
        For i As Integer = 0 To NumberRecordNotifications - 1
            PositionNotify(i).Offset = (NotifySize * i) + NotifySize - 1
            PositionNotify(i).EventNotifyHandle =
NotificationEvent.SafeWaitHandle.DangerousGetHandle()
        Next

        applicationNotify = New Notify(applicationBuffer)

        ' Tell DirectSound when to notify the app. The notification will come in
the from
        ' of signaled events that are handled in the notify thread.
        applicationNotify.SetNotificationPositions(PositionNotify,
NumberRecordNotifications)
    End Sub

    Private Sub WaitThread()
        MsgBox("Start WaitThread")
        While Capturing = True
            'Calling WaitOne stops and waits for the next offset to be reached
            NotificationEvent.WaitOne(Timeout.Infinite, True)
            RecordCapturedData()
        End While
        MsgBox("Stopped Recording")
    End Sub

    Private Sub ButtonFileName_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ButtonFileName.Click

        If SaveFileDialog1.ShowDialog Then
            FileName = SaveFileDialog1.FileName
            Try
                WaveFile = New FileStream(FileName, FileMode.Create,
FileAccess.Write)
            Catch ex As SystemException
```

```
                MsgBox(ex.Message)
                Return
            End Try

        End If
        ButtonFileName.Enabled = False
        ButtonStart.Enabled = True

    End Sub

    Private Sub ButtonStart_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ButtonStart.Click

        ButtonStart.Enabled = False
        ButtonStop.Enabled = True
        CreateSoundFile()
        Me.Text = "Now recording ..."
    End Sub

    Private Sub ButtonStop_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ButtonStop.Click
        ' Stop the buffer, and read any data that was not
        ' caught by a notification

        ButtonStop.Enabled = False
        applicationBuffer.[Stop]()

        RecordCapturedData()

        Writer.Seek(4, SeekOrigin.Begin)
        ' Seek to the length descriptor of the RIFF file.
        Writer.Write(CInt(SampleCount + 36))
        ' Write the file length, minus first 8 bytes of RIFF description.
        Writer.Seek(40, SeekOrigin.Begin)
        ' Seek to the data length descriptor of the RIFF file.
        Writer.Write(SampleCount)
        ' Write the length of the sample data in bytes.
        Writer.Close()
        ' Close the file now.
        Writer = Nothing
        '<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>
        '------------------------------------
        ' Set the writer to null.
        ' Set the FileStream to null.
        WaveFile = Nothing
        Capturing = False
        NotificationEvent.Set()
        Me.Text = "Stopped recording!"
    End Sub
    Private Sub ButtonExit_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ButtonExit.Click

        NotificationEvent.Set()
        MDCT.TextBox1.Text = FileName 'MDCT.Show()
        Me.Close()

    End Sub
```

```
    Private Sub ComboBoxDevice_SelectedIndexChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles ComboBoxDevice.SelectedIndexChanged
        'MsgBox(Convert.ToString(ComboBoxDevice.SelectedIndex))
        'If ComboBoxDevice.SelectedIndex < 0 Then
        '    MsgBox("Please select device from the combobox")
        '    Exit Sub
        If ComboBoxDevice.Items.Count >= 0 Then
            Me.CaptureDeviceGuid =
Devices(ComboBoxDevice.SelectedIndex).DriverGuid
        End If
        CaptureBufferSize = 0
        Try
            ApplicationDevice = New Capture(CaptureDeviceGuid)
            MsgBox("Device ready!")
            'ButtonDevice.Enabled = False
        Catch ex As Exception
            MsgBox(ex.ToString)
        End Try

        ButtonFileName.Enabled = True
    End Sub
    Private Sub StartOrStopRecord(ByVal StartRecording As Boolean)
        '-----------------------------------------------------------------------
----
        ' Name: StartOrStopRecord()
        ' Desc: Starts or stops the capture buffer from recording
        '-----------------------------------------------------------------------
----
        Writer = New BinaryWriter(WaveFile)

        If Recording = StartRecording Then
            ' Create a capture buffer, and tell the capture
            ' buffer to start recording
            CreateCaptureBuffer()
            Recording = True
            applicationBuffer.Start(True)

        Else
            ' Stop the buffer, and read any data that was not
            ' caught by a notification
            applicationBuffer.[Stop]()

            RecordCapturedData()

            Writer.Seek(4, SeekOrigin.Begin)
            ' Seek to the length descriptor of the RIFF file.
            Writer.Write(CInt(SampleCount + 36))
            ' Write the file length, minus first 8 bytes of RIFF description.
            Writer.Seek(40, SeekOrigin.Begin)
            ' Seek to the data length descriptor of the RIFF file.
            Writer.Write(SampleCount)
            ' Write the length of the sample data in bytes.
            Writer.Close()
            ' Close the file now.
            Writer = Nothing
            '<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>
            '--------------------------------------
            ' Set the writer to null.
```

```
            ' Set the FileStream to null.
            WaveFile = Nothing
            'Recording = False
            Capturing = False
            'NotificationEvent.Set()
        End If
    End Sub
    Private Sub CreateRIFF()
        Writer = New BinaryWriter(WaveFile)

        ' Set up file with RIFF chunk info.
        Dim ChunkRiff As Char() = {"R"c, "I"c, "F"c, "F"c}
        Dim ChunkType As Char() = {"W"c, "A"c, "V"c, "E"c}
        Dim ChunkFmt As Char() = {"f"c, "m"c, "t"c, " "c}
        Dim ChunkData As Char() = {"d"c, "a"c, "t"c, "a"c}

        Dim shPad As Short = 1
        ' File padding
        Dim nFormatChunkLength As Integer = &H10
        ' Format chunk length.
        Dim nLength As Integer = 0
        ' File length, minus first 8 bytes of RIFF description. This will be
filled in later.
        Dim shBytesPerSample As Short = 2
        ' Bytes per sample.
        ' Figure out how many bytes there will be per sample.
        If 8 = InputFormat.BitsPerSample AndAlso 1 = InputFormat.Channels Then
            shBytesPerSample = 1
        ElseIf (8 = InputFormat.BitsPerSample AndAlso 2 = InputFormat.Channels)
OrElse (16 = InputFormat.BitsPerSample AndAlso 1 = InputFormat.Channels) Then
            shBytesPerSample = 2
        ElseIf 16 = InputFormat.BitsPerSample AndAlso 2 = InputFormat.Channels
Then
            shBytesPerSample = 4
        End If

        ' Fill in the riff info for the wave file.
        Writer.Write(ChunkRiff)
        Writer.Write(nLength)
        Writer.Write(ChunkType)

        ' Fill in the format info for the wave file.
        Writer.Write(ChunkFmt)
        Writer.Write(nFormatChunkLength)
        Writer.Write(shPad)
        Writer.Write(InputFormat.Channels) 'channels
        Writer.Write(InputFormat.SamplesPerSecond) 'samplepersecond
        Writer.Write(InputFormat.AverageBytesPerSecond) 'averagebytespersecond
        Writer.Write(shBytesPerSample) 'bytespersample
        Writer.Write(InputFormat.BitsPerSample) 'bitspersample

        ' Now fill in the data chunk.
        Writer.Write(ChunkData)
        Writer.Write(CInt(0))
        ' The sample length will be written in later.

'<<
```

```
    End Sub
    Private Sub CreateSoundFile()
        '------------------------------------------------------------------------
----
        ' Name: CreateSoundFile()
        ' Desc: Called when the user requests to save to a sound file
        '------------------------------------------------------------------------
----

        StartOrStopRecord(False)

        Try
            CreateRIFF()
        Catch

        End Try

        ' Update the UI controls to show the sound as the file is loaded
        labelFilename.Text = FileName
        'checkboxRecord.Enabled = True

        ' Remember the path for next time
        Path = FileName.Substring(0, FileName.LastIndexOf("\"))
    End Sub
    Private Sub RecordCapturedData()
        '------------------------------------------------------------------------
----
        ' Name: RecordCapturedData()
        ' Desc: Copies data from the capture buffer to the output buffer
        '------------------------------------------------------------------------
----
        Dim CaptureData As Byte() = Nothing
        Dim ReadPos As Integer
        Dim CapturePos As Integer
        Dim LockSize As Integer

        applicationBuffer.GetCurrentPosition(CapturePos, ReadPos)

        LockSize = ReadPos - NextCaptureOffset
        If LockSize < 0 Then
            LockSize += CaptureBufferSize
        End If

        ' Block align lock size so that we are always write on a boundary
        LockSize -= (LockSize Mod NotifySize)

        If 0 = LockSize Then
            Return
        End If

        ' Read the capture buffer.
        CaptureData = CType(applicationBuffer.Read(NextCaptureOffset,
GetType(Byte), LockFlag.None, LockSize), Byte())

        ' Write the data into the wav file
        Writer.Write(CaptureData, 0, CaptureData.Length)

        ' Update the number of samples, in bytes, of the file so far.
```

```
        SampleCount += CaptureData.Length

        ' Move the capture offset along
        NextCaptureOffset += CaptureData.Length
        NextCaptureOffset = NextCaptureOffset Mod CaptureBufferSize ' Circular
buffer
    End Sub

End Class


--------------------------------------------------------------------------------

Imports System
Imports System.IO
Imports System.Threading
Imports System.Text


Public Class Form3
    '\\ API declarations...
    Private Declare Sub CopyMemory Lib "kernel32" Alias "RtlMoveMemory" (ByVal
Destination As Long, ByVal Source As Long, ByVal Length As Long)
    Private Declare Sub CopyMemoryByte Lib "kernel32" Alias "RtlMoveMemory" (ByVal
Destination As Byte, ByVal Source As Long, ByVal Length As Long)
    'Private Declare Sub CopyMemoryFromByte Lib "kernel32" Alias "RtlMoveMemory"
(ByVal Destination As Long, ByVal Source As Byte, ByVal Length As Long)
    Public Function LenB(ByVal value As String) As Integer
        If value Is Nothing Then Return 0
        Return value.Length * 2
    End Function
    Public Function Peek(ByVal Address As Long) As Long

        Call CopyMemory(Peek, Address, Len(Address))

    End Function

    Public Function PeekByte(ByVal Address As Long) As Byte

        Call CopyMemoryByte(PeekByte, Address, Len(PeekByte))

    End Function

    Shared c As Double() = {-0.6, -0.535, -0.33, -0.185, -0.095, -0.041, -0.0142,
-0.0037}
    Shared ca(8) As Double
    Shared cs(8) As Double
    Shared win(36) As Double
    Shared cos_l(18, 36) As Double
    Const HAN_SIZE = 512
    Public startTime As New DateTime
    Public EndTime As New DateTime
    Public timelength() As Double = New Double(100000) {}
    Private n As Long = 0
    Const BUFSIZE = 4096
    Const BUFFER_SIZE = 4096
    Const SBLIMIT = 32
```

```
    Public offset As ULong
    Public totbit As ULong = 0
    Public buf_byte_idx As ULong = 0
    Public buf(BUFSIZE) As UInteger
    Public buf_bit_idx As UInteger = 8
    Const samp_per_frame = 1152
    Const samp_per_frame2 = 576
    Const MODE_MONO = 3
    Const HTN = 34
    Const PI4 = 0.78539816339745
    Const PI12 = 0.26179938779915
    Const PI36 = 0.087266462599717
    Const PI64 = 0.049087385212
    Const SCALED = 32768
    Const MXOFF = 250
    Public filter(SBLIMIT, 64) As Double 'Static
    Dim RIFFfile As FileStream
    Dim RIFFLength As Long
    Dim RIFFType As String
    Private MP3HeaderBits As BitArray
    Public Textpath As String = "C:\Users\ayotui.MANAGEMENT\Desktop\MP3
CPP\SHINE\yotui\yotui.txt"
    Structure sfBandIndex_t

        Dim l() As Integer
        Dim s() As Integer
    End Structure
    ' Public Property HUFFBITS() As ULong
    'Shared t1HB() As HUFFBITS = {1, 1, 1, 0}
    Structure huffcodetab
        Dim xlen As UInteger   '/*max. x-index+                    */
        Dim ylen As UInteger  '/*max. y-index+                     */
        Dim linbits As UInteger '/*number of linbits               */
        Dim linmax As UInteger    '/*max number to be stored in linbits     */
        'Dim table As HUFFBITS '*table;  '/*pointer to array[xlen][ylen]
       */
        Dim hlena As Byte 'unsigned char'/*pointer to array[xlen][ylen]
       */
    End Structure

    Structure gr_info 'GrInfo
        Dim part2_3_length As UInteger
        Dim big_values As UInteger
        Dim count1 As UInteger
        Dim global_gain As UInteger
        Dim scalefac_compress As UInteger
        Dim table_select() As UInteger
        '  later,  ReDim struct.table_select(3)
        Dim region0_count As UInteger
        Dim region1_count As UInteger
        Dim preflag As UInteger
        Dim scalefac_scale As UInteger
        Dim count1table_select As UInteger
        Dim part2_length As UInteger
        Dim sfb_lmax As UInteger
        Dim address1 As UInteger
        Dim address2 As UInteger
        Dim address3 As UInteger
```

```
        Dim quantizerStepSize As Double
        Dim slen() As UInteger
        'later redim slen(4)
    End Structure
    ' Public gr_info As GrInfo

    Structure side_info_t 'SideInfo
        Dim main_data_begin As Integer ' unsigned -> int
        Dim private_bits As UInteger
        Dim resvDrain As Integer
        Dim scfsi(,) As UInteger
        'later redim scfsi(2,4)
        Structure gr 'G_R
            'later redim gr(2)
            Dim anything As Integer
            Structure ch 'C_H
                'later redim sc(2)
                Dim tt As gr_info
            End Structure
        End Structure
    End Structure

    Structure psy_ratio_t 'PsyRatio
        Dim l(,,) As Double
        'later redim l(2, 2, 21)
    End Structure ' ;

    Structure psy_xmin_t 'PsyMin
        Dim l(,,) As Double
        'later redim l(2, 2, 21)
    End Structure

    Structure Scalefact
        Dim l(,,) As Integer            ' [cb] */
        'later redim l(2, 2, 22)
        Dim s(,,,) As Integer         '[window][cb] */
        'later redim s(2, 2,13,3)
    End Structure ' scalefac_t;
    'Public scalefac_t As Scalefact
    'Public psy_xmin_t As PsyMin
    'Public psy_ratio_t As PsyRatio
    'Public side_info_t As SideInfo
    Structure bitstream_t 'bit_stream_struc
        Dim pt() As File        '*pt; pointer to bit stream device */
        Dim buf() As Char         ' /* bit stream buffer */
        Dim buf_size As Integer         ';       /* size of buffer (in number of
bytes) */
        Dim totbit As Long        ';         /* bit counter of bit stream */
        Dim buf_byte_idx As Integer ';   /* pointer to top byte in buffer */
        Dim buf_bit_idx As Integer ';    /* pointer to top bit of top byte in
buffer */
        Dim mode As Integer ';           /* bit stream open in read or write mode
*/
        Dim eob As Integer ';            /* end of buffer index */
        Dim eobs As Integer ';           /* end of bit stream flag */
        Dim format As Char        ';

        ' format of file in rd mode (BINARY/ASCII)
```

```
End Structure
Enum Important
    byte_order
    e_byte_order
End Enum
Public Structure mpeg_t 'for_mpeg
    Dim file() As File
    Dim type As Integer
    Dim layr As Integer
    Dim mode As Integer
    Dim bitr As Integer
    Dim psyc As Integer
    Dim emph As Integer
    Dim padding As Integer
    Dim samples_per_frame As Long
    Dim bits_per_frame As Long
    Dim bits_per_slot As Long
    Dim total_frames As Long
    Dim bitrate_index As Integer
    Dim samplerate_index As Integer
    Dim crc As Integer
    Dim ext As Integer
    Dim mode_ext As Integer
    Dim copyright As Integer
    Dim original As Integer
End Structure
'Public mpeg_t As for_mpeg
Public Structure wave_t 'The_wave
    Dim file As FileStream
    Dim type As Integer
    Dim channels As Integer
    Dim bits As Integer
    Dim samplerate As Long
    Dim total_samples As Long
    Dim length As Long
End Structure
'Public wave_t As The_wave
Public Structure config_t
    Dim start_time As DateTime 'time_t
    Dim myenum As Important
    Dim infile As Char()
    Dim wave As wave_t
    Dim outfile As Char()
    Dim mpeg As mpeg_t
End Structure
Shared config As config_t

Private Sub mdct_initialise()
    Dim i, m, k As Integer 'N;
    Dim sq As Double
    ' prepare the aliasing reduction butterflies
    For i = 7 To 0 Step -1
        sq = Math.Sqrt(1.0 + Math.Pow(c(i), 2))
        ca(i) = c(i) / sq
        cs(i) = 1.0 / sq
    Next i

    For i = 35 To 0 Step -1
```

```
            win(i) = Math.Sin(Math.PI * 36 * (i + 0.5))
        Next i


        'N = 36;
        For m = 0 To 17
            For k = 0 To 35
                cos_l(m, k) = Math.Cos((Math.PI / (72)) * (2 * k + 19) * (2 * m +
1)) / (9)
            Next k
        Next m

    End Sub

    Private Sub mdct(ByVal SamplesIn() As Double, ByVal SamplesOut() As Double)
        '/*-------------------------------------------------------------------*/
        '/*   Function: Calculation of the MDCT                               */
        '/*   In the case of long blocks ( block_type 0,1,3 ) there are       */
        '/*   36 coefficents in the time domain and 18 in the frequency       */
        '/*   domain.                                                         */
        '/*-------------------------------------------------------------------*/
        Dim k, m As Integer

        For m = 18 To 0 Step -1

            SamplesOut(m) = win(35) * SamplesIn(35) * cos_l(m, 35)
            For k = 35 To 0 Step -1
                SamplesOut(m) += win(k) * SamplesIn(k) * cos_l(m, k)
            Next k
        Next m
    End Sub


    Private Sub mdct_sub(ByVal sb_sample(,,,) As Double, ByRef mdct_freq(,) As
Double, _
                ByRef side_info() As side_info_t)

        ReDim sb_sample(2, 3, 18, SBLIMIT)
        ReDim mdct_freq(2, samp_per_frame2)
        Dim mdct_enc(2, 32, 18) As Double
        'mdct_enc(2, 32, 18) = mdct_freq(2, 32, 18)
        Dim ch, gr, band, k, j As Integer
        Dim cod_info() As gr_info
        Dim mdct_in(36) As Double
        Dim bu, bd As Double

        For gr = 0 To gr < 2
            For ch = config.wave.channels To 0 Step -1

                'cod_info = (gr_info*) &(side_info.gr(gr).ch(ch))

                ' Compensate for inversion in the analysis filter
                For band = 32 To 0 Step -1
                    For k = 18 To 0 Step -1
                        If ((band And 1) And (k And 1)) Then
                            sb_sample(ch, gr + 1, k, band) *= -1.0
                        End If
                    Next k
```

```
                Next band

                ' Perform imdct of 18 previous subband samples + 18 current
subband samples
                For band = 32 To 0 Step -1

                    For k = 18 To 0 Step -1

                        mdct_in(k) = sb_sample(ch, gr, k, band)
                        mdct_in(k + 18) = sb_sample(ch, gr + 1, k, band)
                    Next k

                    'mdct(mdct_in, mdct_enc(gr, ch, band), 0)
                Next band

                ' Perform aliasing reduction butterfly
                For band = 31 To 0 Step -1
                    For k = 8 To 0 Step -1

                        'bu = mdct_enc(gr, ch, band, 17 - k) * cs(k) +
mdct_enc(gr, ch, band + 1, k) * ca(k)
                        'bd = mdct_enc(gr, ch, band + 1, k) * cs(k) - mdct_enc(gr,
ch, band, 17 - k) * ca(k)
                        'mdct_enc(gr, ch, band, 17 - k) = bu
                        'mdct_enc(gr, ch, band + 1, k) = bd
                    Next k
                Next band
            Next ch
        Next gr
        ' Save latest granule's subband samples to be used in the next mdct call
        For ch = config.wave.channels To 0 Step -1
            For j = 18 To 0 Step -1
                For band = 32 To 0 Step -1
                    sb_sample(ch, 0, j, band) = sb_sample(ch, 2, j, band)
                Next band
            Next j
        Next ch
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Dim end_time As DateTime
        Try
            startTime = DateTime.Now 'time((config.start_time))
            print_header()
            set_defaults()
            ''If (Not parse_command(sender, e)) Then
            print_usage()
            ''Return -1
            ''End If

            If Not wave_open() Then
                MsgBox("Unable to open input file...")
            End If

            check_config()

            'MsgBox("Encoding \" & config.infile & " to \" & config.outfile & "")
```

```
            Compress()

        Catch Ex As Exception
            'Throw Ex
            MsgBox(Ex.Message)
        End Try
    End Sub
    Shared Sub update_status(ByVal frames_processed As Integer)
    End Sub
    Public Sub check_config()

        Static mode_names() As Char = {"stereo", "j-stereo", "dual-ch", "mono"}
        Static layer_names() As Char = {"I", "II", "III"}
        Static version_names() As Char = {"MPEG-II (LSF)", "MPEG-I"}
        Static psy_names() As Char = {"", "MUSICAM", "Shine"}
        Static demp_names() As Char = {"none", "50/15us", "", "CITT"}

        MsgBox("" & version_names(config.mpeg.type) & " layer " &
layer_names(config.mpeg.layr) & " , " & mode_names(config.mpeg.mode) & "
Psychoacoustic Model: " & psy_names(config.mpeg.psyc) & "")

        MsgBox("Bitrate= " & config.mpeg.bitr & " kbps  ")

    End Sub
    Shared Function find_samplerate_index(ByVal freq As Long) As Integer

        Static mpeg1() As Long = {44100, 48000, 32000}
        mpeg1(0) = 44100
        mpeg1(1) = 48000
        mpeg1(2) = 32000
        Dim i As Integer
        Dim found As Boolean = False

        For i = 0 To i < 3
            If (freq = mpeg1(i)) Then
                found = True
                MsgBox("sample rate is " & mpeg1(i) & "")
                Return i
            End If

        Next i
        If found = False Then
            MsgBox("Invalid samplerate")
        End If
        'Return -1
    End Function

    Shared Function find_bitrate_index(ByVal bitr As Integer) As Integer
        Static Dim mpeg1() As Long = {0, 32, 40, 48, 56, 64, 80, 96, 112, 128,
160, 192, 224, 256, 320}
        Dim i As Integer

        For i = 0 To i < 15
            If (bitr = mpeg1(i)) Then
                Return i
            End If
        Next i
```

```
        MsgBox("Invalid bitrate")
        Return -1
    End Function
    Private Sub Compress()
        Dim frames_processed As Integer
        Static Dim buffer(2, samp_per_frame) As Short
        Dim channel As Integer

        Dim i As Integer
        Dim gr As Integer
        Dim sam(2, 1344) As Short
        Dim snr32(32) As Double
        Dim side_info As side_info_t
        Dim buffer_window(2) As Short
        Dim win_que(2, HAN_SIZE) As Double
        Dim sb_sample(2, 3, 18, SBLIMIT) As Double
        Dim mdct_freq(2, 2, samp_per_frame2) As Double
        Dim enc(2, 2, samp_per_frame2) As Integer
        Dim scalefactor As Scalefact
        Dim bs As bitstream_t

        Dim avg_slots_per_frame As Double
        Dim frac_slots_per_frame As Double
        Dim whole_slots_per_frame As Long
        Dim slot_lag As Double

        Dim mean_bits As Integer
        Dim sent_bits As Long = 0
        Dim frame_bits As Long = 0
        Dim sideinfo_len As Integer
        Dim OutputX As FileStream = Nothing
        Dim datafile As String = Nothing
        Try
            'open_bit_stream_w(&bs, config.outfile, BUFFER_SIZE)

            subband_initialise()
            mdct_initialise()

            config.mpeg.samples_per_frame = samp_per_frame
            config.mpeg.total_frames = config.wave.total_samples /
config.mpeg.samples_per_frame
            config.mpeg.bits_per_slot = 8
            frames_processed = 0
            sideinfo_len = 32

            If (config.wave.channels = 1) Then
                sideinfo_len += 136
            Else
                sideinfo_len += 256
            End If

            '' Figure average number of 'slots' per frame.
            avg_slots_per_frame = (Convert.ToDouble(config.mpeg.samples_per_frame)
/ (Convert.ToDouble(config.wave.samplerate) / 1000) *
(Convert.ToDouble(config.mpeg.bitr) / _
                                   Convert.ToDouble(config.mpeg.bits_per_slot)))
            'MsgBox(config.mpeg.bits_per_slot)
```

```
            whole_slots_per_frame = CInt(avg_slots_per_frame)
            frac_slots_per_frame = avg_slots_per_frame -
Convert.ToDouble(whole_slots_per_frame)
            slot_lag = -frac_slots_per_frame
            If (frac_slots_per_frame = 0) Then
                config.mpeg.padding = 0
            End If

            While (wave_get(buffer))

                If ((frames_processed + 1) Mod 7 = 0) Then
                    update_status(frames_processed)
                End If

                buffer_window(0) = buffer(0, samp_per_frame)
                buffer_window(1) = buffer(1, samp_per_frame)

                If (frac_slots_per_frame) Then
                    If (slot_lag > (frac_slots_per_frame - 1.0)) Then
                        ' No padding for this frame
                        slot_lag -= frac_slots_per_frame
                        config.mpeg.padding = 0
                    End If
                Else
                    ' Padding for this frame
                    slot_lag += (1 - frac_slots_per_frame)
                    config.mpeg.padding = 1
                End If
                config.mpeg.bits_per_frame = 8 * (whole_slots_per_frame +
config.mpeg.padding)
                mean_bits = (config.mpeg.bits_per_frame - sideinfo_len) >> 1


                ' polyphase filtering
                For gr = 0 To gr < 2
                    For channel = config.wave.channels To 0 Step -1
                        For i = 0 To i < 18

                            window_subband(buffer_window(channel),
win_que(channel, 0), channel)
                            filter_subband(win_que(channel, 0), sb_sample(channel,
gr + 1, i, 0))
                        Next i
                    Next channel
                Next gr

                sent_bits += frame_bits
            End While
        Catch Ex As Exception
            Throw Ex
            MsgBox(Ex.Message)
        End Try

    End Sub
    Private Sub iteration_loop(ByVal mdct_freq_org(,,) As Double, ByRef side_info
As side_info_t, ByVal enc(,,) As Integer, ByVal mean_bits As Integer, ByRef
scalefactor As Scalefact)
```

```
'/************************************************************************/
        'mdct_freq_org[2][2][samp_per_frame2]
        'enc[2][2][samp_per_frame2]
        '/*  iteration_loop()
*/

'/************************************************************************/

        Dim cod_info As gr_info
        Dim main_data_begin As Integer
        'Dim scalefac_band_long = Peek(sfBandIndex(3).l(0))
        Dim max_bits As Integer
        Dim ch, gr, i As Integer
        Static firstcall As Integer = 1

        Dim xr(2, 2, samp_per_frame2) As Double
        'For i = 0 To 5
        'Dim l(23) As Integer
        'Dim s(14) As Integer
        '    sfBandIndex(i).l()
        '    sfBandIndex(i).s()
        'Next
        main_data_begin = Peek(side_info.main_data_begin)

        If (firstcall) Then

            main_data_begin = 0
            firstcall = 0
        End If

        'scalefac_band_long = sfBandIndex(config.mpeg.samplerate_index +
(config.mpeg.type * 3)).l(0)


        For gr = 2 To 0 Step -1

            For ch = config.wave.channels To 0 Step -1

                For i = samp_per_frame2 To 0 Step -1
                    xr(gr, ch, i) = mdct_freq_org(gr, ch, i)
                Next i
            Next ch
        Next gr



        For gr = 2 To 0 Step -1

            For ch = config.wave.channels To 0 Step -1

                'cod_info = Peek(gr_info) And side_info.gr(gr).ch(ch)

                '/* calculation of number of available bit( per granule ) */
                max_bits = mean_bits / config.wave.channels

                '/* reset of iteration variables */
                'memset(scalefactor.l(gr, ch), 0, 22)
```

```
                'memset(scalefactor.s(gr, ch), 0, 14)


                For i = 4 To 0 Step -1
                    cod_info.slen(i) = 0
                Next i
                cod_info.part2_3_length = 0
                cod_info.big_values = 0
                cod_info.count1 = 0
                cod_info.scalefac_compress = 0
                cod_info.table_select(0) = 0
                cod_info.table_select(1) = 0
                cod_info.table_select(2) = 0
                cod_info.region0_count = 0
                cod_info.region1_count = 0
                cod_info.part2_length = 0
                cod_info.preflag = 0
                cod_info.scalefac_scale = 0
                cod_info.quantizerStepSize = 0.0
                cod_info.count1table_select = 0

                'cod_info.quantizerStepSize =
Convert.ToDouble(quantanf_init(xr(gr, ch, samp_per_frame2)))
                cod_info.part2_3_length = outer_loop(xr, max_bits, enc, gr, ch,
side_info)
                ResvAdjust(cod_info, mean_bits)

                cod_info.global_gain = cod_info.quantizerStepSize + 210

            Next ch '/* for ch */
        Next gr '/* for gr */

        ResvFrameEnd(side_info)
    End Sub

    '/*****************************************************************/
    '/*  quantanf_init                                                */
    '/*****************************************************************/
    Function quantanf_init(ByVal xr() As Double) As Integer
        '/* Function: Calculate the first quantization step quantanf.       */
        'xr(samp_per_frame2)

        Dim i, tp As Integer
        Dim system_const As Double = 8
        Dim sfm As Double = 0
        Dim sum1 As Double = 0
        Dim sum2 As Double = 0


        For i = samp_per_frame2 To 0 Step -1
            If xr(i) Then

                Dim tpd As Double = xr(i) * xr(i)
                sum1 += Math.Log(tpd)
                sum2 += tpd
            End If
        Next
```

```
        If sum2 Then

            sfm = Math.Exp(sum1 / samp_per_frame2) / (sum2 / samp_per_frame2)
            tp = CInt(system_const * Math.Log(sfm))
            If (tp < -100) Then
                tp = -100
            End If
        End If


        Return (tp - 70) '/* SS 19-12-96. Starting value of
        '   global_gain or quantizerStepSize
        '   has to be reduced for iteration_loop

    End Function
    Shared Function outer_loop(ByVal xr(,,) As Double, _
                       ByVal max_bits As Integer, ByVal enc(,,) As Integer, _
                       ByVal gr As Integer, ByVal ch As Integer, ByRef side_info
As side_info_t) As Integer

'/*************************************************************************/
        '/*  Function: The outer iteration loop controls the masking conditions
*/
        '/*  of all scalefactorbands. It computes the best scalefac and
*/
        '/*  global gain. This module calls the inner iteration loop
*/\xr(2,2,samp_per_frame2)

'/*************************************************************************/enc(2,2
,samp_per_frame2)
        Dim bits, huff_bits As Integer

        Dim xrs As Double
        Dim ix As IntPtr
        Dim cod_info As gr_info '= side_info.gr(gr).ch(ch).tt

        Return cod_info.part2_3_length
    End Function
    Function bin_search_StepSize(ByVal desired_rate As Integer, ByVal start As
Double, ByVal ix() As IntPtr, _
                         ByVal xrs() As Double, ByRef cod_info As gr_info) As
Integer
        'xrs(samp_per_frame2)
        Dim top, bot, Thenext, last As Integer
        Dim bit As Integer

        top = start
        Thenext = start
        bot = 200

        Do

            last = Thenext
            Thenext = (CLng(top + bot) >> 1)

            cod_info.quantizerStepSize = Thenext
            quantize(xrs, ix, cod_info)
            bit = count_bits(ix, cod_info)
```

```
            If (bit > desired_rate) Then
                top = Thenext
            Else
                bot = Thenext
            End If
        Loop While ((bit <> desired_rate) And Math.Abs(last - Thenext) > 1)

        Return Thenext
    End Function
    Function count_bits(ByVal ix() As IntPtr, ByRef cod_info As gr_info) As
Integer

        Dim bits, max As Integer

    End Function
    Private Sub subdivide(ByRef cod_info As gr_info)

'/*************************************************************************/
        '/* presumable subdivides the bigvalue region which will use separate
Huffman tables.  */

'/*************************************************************************/

        Dim scfb_anz As Integer = 0
        Dim bigvalues_region As Integer

        If Not cod_info.big_values Then
            ' no big_values region */
            cod_info.region0_count = 0
            cod_info.region1_count = 0
        Else

            bigvalues_region = 2 * cod_info.big_values


            Dim thiscount, index As Integer
        End If

    End Sub


    Private Sub bigv_tab_select(ByVal ix() As Integer, ByRef cod_info As gr_info)

'/*************************************************************************/
        '/*   Function: Select huffman code tables for bigvalues regions */

'/*************************************************************************/
        'ix(samp_per_frame2)

        cod_info.table_select(0) = 0
        cod_info.table_select(1) = 0
        cod_info.table_select(2) = 0


        If cod_info.address1 > 0 Then
            cod_info.table_select(0) = new_choose_table(ix, 0, cod_info.address1)
        End If
```

```
        If cod_info.address2 > cod_info.address1 Then
            cod_info.table_select(1) = new_choose_table(ix, cod_info.address1,
cod_info.address2)
        End If
        If (cod_info.big_values << 1 > cod_info.address2) Then
            cod_info.table_select(2) = new_choose_table(ix, cod_info.address2,
cod_info.big_values << 1)
        End If
    End Sub
    Function new_choose_table(ByVal ix() As Integer, ByVal begin As UInteger,
ByVal ends As UInteger) As Integer

'/*************************************************************************/
        '/*  Choose the Huffman table that will encode ix[begin..end] with
*/
        '/*  the fewest bits.
*/
      '/*************************************************************************/
        'ix(samp_per_frame2)
        Dim i, max As Integer
        Dim choice(2) As Integer
        Dim sum(2) As Integer

        max = ix_max(ix, begin, ends)
        If (Not max) Then
            Return 0
        End If
        choice(0) = 0
        choice(1) = 0

        If (max < 15) Then

            ' try tables with no linbits */
            For i = 14 To 0 Step -1
                'If ht(i).xlen > max Then

                '    choice(0) = i
                '    Exit For
                'End If
                'End If
            Next i
            sum(0) = count_bit(ix, begin, ends, choice(0))

            Select Case choice(0)

                Case 2
                    sum(1) = count_bit(ix, begin, ends, 3)
                    If sum(1) <= sum(0) Then
                        choice(0) = 3
                    End If

                Case 5
                    sum(1) = count_bit(ix, begin, ends, 6)
                    If sum(1) <= sum(0) Then
                        choice(0) = 6
                    End If
                Case 7
                    sum(1) = count_bit(ix, begin, ends, 8)
```

```
            If sum(1) <= sum(0) Then

                choice(0) = 8
                sum(0) = sum(1)
            End If
            sum(1) = count_bit(ix, begin, ends, 9)
            If sum(1) <= sum(0) Then
                choice(0) = 9
            End If

        Case 10
            sum(1) = count_bit(ix, begin, ends, 11)
            If sum(1) <= sum(0) Then

                choice(0) = 11
                sum(0) = sum(1)
            End If
            sum(1) = count_bit(ix, begin, ends, 12)
            If sum(1) <= sum(0) Then
                choice(0) = 12
            End If

        Case 13
            sum(1) = count_bit(ix, begin, ends, 15)
            If sum(1) <= sum(0) Then
                choice(0) = 15
            End If

    End Select
Else

    '/* try tables with linbits */
    max -= 15

    For i = 15 To 23

        'If ht(i).linmax >= max Then

        '    choice(0) = i
        '    Exit For 'break;
        'End If
    Next
    For i = 24 To 31

        'If ht(i).linmax >= max Then

        '    choice(1) = i
        '    Exit For
        'End If
    Next

    sum(0) = count_bit(ix, begin, ends, choice(0))
    sum(1) = count_bit(ix, begin, ends, choice(1))
    If sum(1) < sum(0) Then
        choice(0) = choice(1)
    End If
End If
Return choice(0)
```

```
    End Function
    Function count_bit(ByVal ix() As Integer, ByVal start As UInteger, _
              ByVal ends As UInteger, ByVal table As UInteger) As Integer

'/*************************************************************************/
        '/* Function: Count the number of bits necessary to code the subregion. */

'/*************************************************************************/

        Dim linbits, ylen As UInteger
        Dim i, sum As Integer
        Dim x, y As Integer
        ' struct(huffcodetab * h)

        If Not table Then
            Return 0
        End If

        'h    = &(ht[table]);
        sum = 0

    End Function

    Function count1_bitcount(ByVal ix() As Integer, ByRef cod_info As gr_info) As
Integer

'/*************************************************************************/
        '/* Determines the number of bits to encode the quadruples.
*/

'/*************************************************************************/
        'ix(samp_per_frame2)

        Dim p, i, k As Integer
        Dim v, w, x, y, signbits As Integer
        Dim sum0 As Integer = 0
        Dim sum1 As Integer = 0

        For i = cod_info.big_values << 1 To cod_info.count1 - 1 'i+=4,

            v = Math.Abs(ix(i))
            w = Math.Abs(ix(i + 1))
            x = Math.Abs(ix(i + 2))
            y = Math.Abs(ix(i + 3))

            p = v + (w << 1) + (x << 2) + (y << 3)

            signbits = 0
            If (v <> 0) Then
                signbits += 1
            End If

            If (w <> 0) Then
                signbits += 1
            End If

            If (x <> 0) Then
                signbits += 1
```

```
            End If

            If (y <> 0) Then
                signbits += 1
            End If


            sum0 += signbits
            sum1 += signbits

            'sum0 += ht(32).hlen(p)
            'sum1 += ht(33).hlen(p)
        Next

        If (sum0 < sum1) Then

            cod_info.count1table_select = 0
            Return sum0

        Else
            cod_info.count1table_select = 1
            Return sum1
        End If
    End Function


    Function ix_max(ByVal ix() As Integer, ByVal begin As UInteger, ByVal ends As
UInteger) As Integer

'/************************************************************************/
        '/*  Function: Calculate the maximum of ix from 0 to 575
*/

'/************************************************************************/
        'ix(samp_per_frame2)

        Dim i, x As Integer
        Dim max As Integer = 0

        For i = begin To i < ends

            x = Math.Abs(ix(i))
            If (x > max) Then
                max = x
            End If
        Next
        Return max
    End Function
    Private Sub calc_runlen(ByVal ix() As Integer, ByRef cod_info As gr_info)

'/************************************************************************/
        '/* Function: Calculation of rzero, count1, big_values
*/
        '/* (Partitions ix into big values, quadruples and zeros).
*/

'/************************************************************************/
        'ix(samp_per_frame2)
```

```
        Dim i As Integer
        Dim rzero As Integer = 0

        For i = samp_per_frame2 To 2 Step -2
            If Not ix(i - 1) And Not ix(i - 2) Then
                rzero = rzero + 1
            Else
                Exit Sub
            End If
        Next

        cod_info.count1 = 0
        'For i = samp_per_frame2 To 4 Step -4
        '    'If Math.Abs(ix(i - 1) <= 1 And Math.Abs(ix(i - 2)) <= 1 And
Math.Abs(ix(i - 3)) <= 1 And Math.Abs(ix(i - 4)) <= 1 Then
        '    cod_info.count1 += 1
        '    Else
        '    Exit Sub
        '    End If
        'Next

        cod_info.big_values = i >> 1

    End Sub
    Private Sub quantize(ByVal xr() As Double, ByVal ix() As IntPtr, ByRef
cod_info As gr_info)

'/*****************************************************************************/
        '/*   Function: Quantization of the vector xr ( -> ix)
*/
        'xr(samp_per_frame2) as double, int ix(samp_per_frame2) as integer

'/*****************************************************************************/

        Dim i As Integer
        Dim idx As Integer
        Dim steps As Double
        Dim dbl As Double
        Dim ln As Long

        steps = Math.Pow(2.0, (cod_info.quantizerStepSize) / 4)

        For i = samp_per_frame2 To 0 Step -1

            dbl = Math.Abs(xr(i)) / steps

            ln = CLng(dbl)
            'If (ln < 10000) Then

            '    idx = int2idx(ln)
            '    If (dbl < idx2dbl(idx + 1)) Then
            '        ix(i) = idx
            '    Else
            '        ix(i) = idx + 1
            '    End If
            'End If
```

```
        Next
    End Sub
    Private Sub ResvAdjust(ByRef gi As gr_info, ByVal mean_bits As Integer)
        Static ResvSize As Integer = 0 ' /* in bits */
        ResvSize += (mean_bits / config.wave.channels) - gi.part2_3_length

    End Sub
    Private Sub ResvFrameEnd(ByRef side As side_info_t)
        Static ResvSize As Integer = 0 ' /* in bits */


        Dim gi As gr_info

        'gi = (gr_info *) and side.gr(0).ch(0))
        gi.part2_3_length += ResvSize
        ResvSize = 0
    End Sub



    Shared Function checkString(ByVal file As FileStream, ByRef header As String)
As Boolean

        'char temp[1024];
        'int  length = strlen(string);
        'if(fread(temp,1,length,file)!=length) ERROR("Premature EOF in input
!?!");
        'temp[length] = (char)0;
        'return !strcmp(temp,string);

        Try
            Dim Reader As FileStream = New FileStream(file.Name, FileMode.Open,
FileAccess.Read) ' Create an instance of StreamReader to read from a file.
            'Dim sr As StreamReader = New StreamReader(file.Name)
            Dim line As String
            ' Read and display the lines from the file until the end
            ' of the file is reached.
            '

'=============================================================================
==================
            Dim temp(1024) As Char
            Dim length As Integer = header.Length
            While (True)

                line = Reader.ReadByte

                If line Then
                    temp(length) = Nothing 'Convert.ToChar(0)
                    If Not String.Compare(temp, header) Then
                        Return Not String.Compare(temp, header)
                    End If
                End If

                If line = Nothing Then
                    Exit Function 'break()
                End If
                Seek(1, 3)
```

```
            End While
            Return 0

            Reader.Close()

        Catch E As Exception
            ' Let the user know what went wrong.
            MsgBox("The file could not be read:")
            MsgBox(E.Message)
        End Try
    End Function
    Public Sub print_header()
        ' Get the directories currently on the C drive.
        Using writer As StreamWriter = New StreamWriter(Textpath, True)
            'Using writer As StreamWriter = New StreamWriter(cDirs.", True)

            writer.WriteLine("Shine 0.1.4")
        End Using
    End Sub

    Public Sub print_usage()
        Using writer As StreamWriter = New StreamWriter(Textpath, True)
            'Using writer As StreamWriter = New StreamWriter(cDirs.", True)

            writer.WriteLine("USAGE   :  Shine [options] <infile> <outfile>")
            writer.WriteLine("OPTIONS : -h            this help message ")
            writer.WriteLine("    -b <bitrate>  set the bitrate [32-320], default
128kbit ")
            writer.WriteLine("  -c            set copyright flag, default off ")
            writer.WriteLine("")
        End Using

    End Sub

    Public Sub set_defaults()

        'config.myenum.byte_order = DetermineByteOrder()
        'If (config.byte_order = order_unknown) Then
        '    MsgBox("Can't determine byte order")
        'End If
        config.mpeg.type = 1
        config.mpeg.layr = 2
        config.mpeg.mode = 2
        config.mpeg.bitr = 128
        config.mpeg.psyc = 2
        config.mpeg.emph = 0
        config.mpeg.crc = 0
        config.mpeg.ext = 0
        config.mpeg.mode_ext = 0
        config.mpeg.copyright = 0
        config.mpeg.original = 1
    End Sub

    Function wave_open() As Boolean
        'Static channel_mappings() As Char = {Nothing, "mono", "stereo"}
        'Dim wFormatTag As Short
        'Dim dAvgBytesPerSec As Long
        'Dim wBlockAlign As Short
```

```
        'Dim filesize As Long
        'Dim header_size As Long
        Dim BitConv As BitConverter
        Dim BitRead As BinaryReader
        Dim filename As String = Nothing
        'If OpenFileDialog1.ShowDialog Then
        '    filename = OpenFileDialog1.FileName
        'End If
        Try
            RIFFfile = IO.File.Open(filename, IO.FileMode.Open,
IO.FileAccess.ReadWrite)

            'RIFFType = ReadFourCC()
            'RIFFfile.Seek(0, IO.SeekOrigin.Begin)

            If ReadFourCC() = "RIFF" Then

                MsgBox("It is a riff")
                'SaveFileDialog1.ShowDialog()
                'datafile = SaveFileDialog1.FileName
                'OutputX = New FileStream(datafile, FileMode.Create,
FileAccess.Write)
            Else

                Throw New Exception("Not A Riff")

            End If


            config.wave.channels = 2 'BitRead.ReadUInt16(config.wave.file)
            config.wave.samplerate = 32000 'Read32BitsLowHigh(config.wave.file)
            Return True
        Catch Ex As Exception
            Throw Ex
            MsgBox(Ex.Message)
        End Try
    End Function
    Private Function GetBitsString(ByVal StartIndex As Integer, ByVal EndIndex As
Integer) As String
        Dim BitsString As String
        Dim i As Integer
        For i = StartIndex To EndIndex
            Select Case MP3HeaderBits.Item(i)
                Case True : BitsString += "1"
                Case False : BitsString += "0"
            End Select
        Next
        Return BitsString
    End Function
    Function ReadFourCC() As String

        'read four bytes into a string
        Dim Result As String = ""

        Result = Chr(RIFFfile.ReadByte()) & Chr(RIFFfile.ReadByte()) &
Chr(RIFFfile.ReadByte()) & Chr(RIFFfile.ReadByte())

        Return Result
```

```
    End Function
    Function wave_get(ByVal buffer(,) As Short) As Integer
        ' expects an interleaved 16bit pcm stream from read_samples, which it
        ' de-interleaves into buffer
        ReDim buffer(2, samp_per_frame)
        Static temp_buf(2304) As Short
        Dim samples_read As Integer
        Dim j As Integer

        Select Case (config.mpeg.mode)

            Case MODE_MONO
                'samples_read = read_samples(temp_buf,
config.mpeg.samples_per_frame)
                For j = 0 To j < samp_per_frame

                    buffer(0, j) = temp_buf(j)
                    buffer(1, j) = 0
                Next j

            Case Else
                'samples_read = read_samples(temp_buf,
config.mpeg.samples_per_frame << 1)
                For j = 0 To j < samp_per_frame

                    buffer(0, j) = temp_buf(2 * j)
                    buffer(1, j) = temp_buf(2 * j + 1)
                Next j

        End Select
        Return samples_read
    End Function
    Private Sub subband_initialise()
        Static off() As Integer = {0, 0}
        Static x(2, HAN_SIZE) As Double
        Static filter(SBLIMIT, 64) As Double
        Dim i, j, k As Integer
        Dim whole As Integer
        Dim fract As Double
        For i = 2 To 0 Step -1
            For j = HAN_SIZE To 0 Step -1
                x(i, j) = 0
            Next j
        Next i

        '/* create_ana_filter */
        '/*************************************************************************
        '* PURPOSE:  Calculates the analysis filter bank coefficients
        '* SEMANTICS:
        '* Calculates the analysis filterbank coefficients and rounds to the
        '* 9th decimal place accuracy of the filterbank tables in the ISO
        '* document.  The coefficients are stored in #filter#
        '*************************************************************************/
        For i = 32 To 0 Step -1
            For k = 64 To 0 Step -1
                whole = CInt(filter(i, k) + 0.5)
                fract = filter(i, k) + 0.5 - whole
```

```
                If ((filter(i, k) = 1000000000.0 * Math.Cos(Convert.ToDouble(((2 *
i + 1) * (16 - k) * PI64)))) >= 0) Then
                    'modf(filter(i, k) + 0.5, filter(i, k))
                    filter(i, k) = Fix(filter(i, k) + 0.5)
                Else
                    'modf(filter(i, k) - 0.5, filter(i, k))
                    filter(i, k) = Fix(filter(i, k) - 0.5)
                End If
                filter(i, k) *= 0.000000001
            Next k
        Next i
    End Sub


    Private Sub window_subband(ByRef buffer As Short, ByRef z As Double, ByVal k
As Integer)
        '/**************************************************************************
        '* PURPOSE:  Overlapping window on PCM samples
        '**************************************************************************/
        Dim i As Integer
        Static x(2, HAN_SIZE) As Double
        Static off() As Integer = {0, 0}
        'byval z(HAN_SIZE) as double

        'replace 32 oldest samples with 32 new samples */
        For i = 0 To 31
            'x(k,31-i+off(k)) = convert.ToDouble(buffer)+=1/SCALED
        Next i

        ' shift samples into proper window positions */
        For i = HAN_SIZE To 0 Step -1
            'z(i) = x(k, (i + off(k)) And (HAN_SIZE - 1)) * enwindow(i)
        Next i

        off(k) += 480             ' offset is modulo (HAN_SIZE)*/
        off(k) = off(k) And HAN_SIZE - 1
    End Sub

    Private Sub filter_subband(ByRef z As Double, ByRef s As Double)
        '
/**************************************************************************
        '* PURPOSE:  Calculates the analysis filter bank coefficients
        '* SEMANTICS:
        '*      The windowed samples #z# is filtered by the digital filter matrix
#filter#
        '**************************************************************************/
        Dim y(64) As Double
        Dim i, j As Integer

    End Sub


End Class
```